MASTER EN SCIENCES INFORMATIQUES

SPECTRUM, le nouveau logiciel de gestion du réseau du CERN

Platteau, Laurent

Award date:
1995

Awarding institution:
Universite de Namur

Facultés Universitaires Notre-Dame de la Paix, Namur
Institut d'Informatique

Année académique 1994 - 1995

SPECTRUM, le nouveau logiciel de gestion du réseau du CERN.

Laurent Platteau

Promoteur : Monsieur Philippe van Bastelaer

Mémoire présenté en vue de l'obtention du grade de Licencié et Maître en Informatique.

RESUME:

Depuis plusieurs années, nous assistons à une explosion sans précédent de l'utilisation des réseaux informatiques. Des quantités phénoménales d'informations y transitent quotidiennement. Cette explosion exige une gestion de plus en plus précise et de plus en plus large de ces réseaux, s'étendant à travers le monde et constitués pour la plupart de composants hétérogènes. C'est pour cette raison que l'on a vu apparaître récemment, sur le marché, des logiciels de gestion de réseaux de plus en plus performants capables de gérer ces véritables "toiles d'araignées" mondiales de réseaux.

Dans ce mémoire, nous allons essentiellement parler du logiciel de gestion de réseaux informatiques, SPECTRUM. Ce logiciel, créé par la firme américaine Cabletron, a été acheté par le CERN (Commission Européenne pour la Recherche Nucléaire) dans le but de gérer l'entièreté de son propre réseau. Nous décrirons également l'environnement du CERN dans lequel le logiciel SPECTRUM est amené à tourner ainsi que les projets qu'on nous a demandé de réaliser et qui se rapportent à la phase de développement du logiciel SPECTRUM.
Et nous terminerons ce mémoire par une présentation de deux autres logiciels de gestion de réseaux existant actuellement sur le marché et rencontrant un certain succès auprès des administrateurs de réseaux.

ABSTRACT:

For several years we have observed an unprecedented explosion of networks useage which daily carry phenomenal volumes of information. This explosion of networks which spread throughout the world and are composed mainly of heterogeneous elements, demands more and more precise as well as better and better management. This is why more comprehensive and better performing software that is able to guide users through the World Wide Web, has recently appeared on the market.

In this discourse we will essentially analyze the SPECTRUM network management software. This software, developed by the American company Cabletron, was purchased by CERN (European Commission for Nuclear Research) to manage and "guide" all its network. We will also describe the CERN environment in which SPECTRUM is applied as well as the projects we were asked to realize which pertain to SPECTRUM software development.
We will end this discourse with a brief presentation of two other software products which are presently being marketed for networks management.

REMERCIEMENTS:

Avant de me plonger au coeur du sujet de ce mémoire, je tiens à remercier vivement certaines personnes qui m'ont aidé, de près ou de loin, à sa réalisation.

Plus particulièrement, je désire remercier:

le Professeur Philippe van Bastelaer de l'Institut d'Informatique des Facultés de Namur qui a beaucoup oeuvré pour m'offrir la chance d'effectuer mon stage au CERN à Genève et qui m'a donné de précieux conseils lors de la rédaction de ce mémoire;

le Docteur J.N. Gamble, chef du groupe CS ("Communications Systems") dans laquelle j'ai travaillé durant mon stage, qui a dirigé avec énormément de gentillesse mon travail;

Daniel Davids, membre du groupe CS, qui m'a pris en charge dès mon arrivée au CERN et qui m'a conseillé et aidé tout au long de mon stage;

Monsieur Questiaux qui a pu me confectionner en temps voulu une "adaptation-souris" qui m'était indispensable pour la bonne réalisation de mon stage;

Monsieur Bertrand Frammery, mon parrain, qui m'a si gentiment accueilli chez lui tout au long des six mois passés à Genève;

ainsi que toutes les autres personnes que je n'ai pas mentionnées individuellement, mais qui ont également participé à la bonne réalisation de mon stage et de mon mémoire.

Merci beaucoup,

Laurent PLATTEAU

TABLE DES MATIERES

INTRODUCTION

De nos jours, dans un monde où la "communication d'information" prime, l'utilisation des réseaux informatiques est devenue chose de plus en plus courante et même indispensable.

Ainsi, avec la prolifération sans cesse croissante des réseaux informatiques et de leur inter-opérabilité, la gestion de ces réseaux est devenue cruciale. Il faut veiller d'une part à ce que ces derniers puissent offrir aux utilisateurs des services fiables et adéquats et d'autre part à un transfert rapide de l'information.

Pour rencontrer ces objectifs, diverses grandes firmes internationales d'informatique ont développé des logiciels de gestion de réseaux informatiques. Ces logiciels se chargent principalement de la gestion en temps réel des réseaux, gestion qui englobe la gestion des pannes des composants, la gestion du trafic ainsi que la gestion d'autres événements pouvant survenir sur les réseaux. Toutes ces informations sont analysées par le logiciel et présentées aux divers administrateurs et opérateurs afin qu'ils puissent disposer d'une vue d'ensemble du réseau et remédier au plus vite aux problèmes qui y sont détectés.

Dans ce mémoire, nous nous concentrerons essentiellement sur le logiciel de gestion de réseaux SPECTRUM créé par la société américaine Cabletron et qui a été acheté par le CERN (Commission Européenne pour la Recherche Nucléaire) pour la gestion de son réseau. Nous présenterons et analyserons également l'environnement du CERN dans lequel le logiciel SPECTRUM sera amené à fonctionner et sur lequel nous avons travaillé tout au long de notre stage.

Dans le premier chapitre, nous expliquerons ce qu'est le CERN, pourquoi il a été créé et en quoi consistent les recherches qui y sont poursuivies depuis quarante ans.
Pour traiter tous les résultats provenant de ces diverses expériences, le CERN dispose d'ordinateurs très puissants capables d'exécuter des centaines de millions d'instructions par seconde.
Nous montrerons la nécessité pour le CERN de disposer dès lors d'un réseau informatique bien développé et d'un bon système de gestion de réseaux pour faire transiter toutes ces montagnes d'informations entre les machines sophistiquées et les puissants ordinateurs. Nous donnerons également une brève description de la topologie du réseau du CERN.

Le deuxième chapitre sera consacré aux problèmes liés à la gestion de réseaux en général et à la présentation du logiciel SPECTRUM dans laquelle nous décrirons les principales fonctions offertes par le logiciel. Ensuite, nous énoncerons ses atouts qui lui ont permis d'être choisi parmi l'ensemble des logiciels de gestion de réseaux figurant actuellement sur le marché.
Nous dirons également quelques mots sur le protocole SNMP qui est utilisé par SPECTRUM pour effectuer tous les "pollings" des appareils situés sur le réseau du CERN, ainsi que sur la

M.I.B. ("Management Information Base") dans laquelle sont stockées les variables d'information caractérisant l'état d'un appareil.
Et finalement, nous essaierons de voir dans quelle mesure, le logiciel SPECTRUM peut être considéré comme un logiciel pour la gestion intégrée de réseaux.

Le troisième chapitre s'occupera de situer le cadre des trois projets que nous avons développés au CERN.
Nous décrirons tout d'abord, dans les grandes lignes, le logiciel de gestion de réseaux CAW (Communications Analysis Workstation) qui tourne actuellement au CERN et qui sera amené à être remplacé par SPECTRUM.
Nous présenterons ensuite deux applications importantes développées par les ingénieurs du CERN, utilisées par le logiciel CAW et en rapport direct avec nos deux premiers projets.
La première, le "*Lego-Plot*", donne un aperçu graphique des performances des appareils appartenant au réseau du CERN pour permettre aux dépanneurs d'"anticiper" les pannes réelles.
La deuxième, "*AlarmMirror*", s'occupe d'afficher sur l'écran de l'opérateur toutes les alarmes provenant des appareils du réseau et détectées par le logiciel CAW. Nous expliquerons également l'utilité et les fonctions d'un système d'alarme.

Les trois chapitres suivants auront chacun trait à l'explication d'un de nos projets.
Le premier projet, "LectureDonnees", s'occupe de créer un fichier lisible par l'application "*Lego-Plot*" à partir des données sur les performances des appareils du réseau, contenues dans la base de données du logiciel SPECTRUM.
Le deuxième projet, "Alarm_Mirror", se charge d'envoyer à l'application "*AlarmMirror*" toutes les alarmes provenant d'appareils du réseau détectées par SPECTRUM.
Le troisième et dernier projet, "Location_View", se rapporte à la configuration des vues "*Localisation*" de SPECTRUM. Il se charge de copier toutes les icônes symbolisant des appareils qui se situent dans la vue "*Topologie*" de SPECTRUM vers les vues "*Localisation*" correspondant aux réels emplacements physiques du CERN (le bâtiment ou la salle) où se trouvent les appareils en question.
Pour chacun de ces trois projets, nous expliquerons leur utilité, la façon dont ils ont été conçus ainsi que les problèmes rencontrés lors de leur conception.

Dans le septième chapitre, nous donnerons une brève présentation de deux autres logiciels de gestion de réseaux qui se trouvent actuellement sur le marché. Nous tenterons également de réaliser quelques comparaisons avec le logiciel SPECTRUM.

Et enfin, nous terminerons ce mémoire par une brève conclusion dans laquelle nous présenterons le bénéfice très constructif que nos six mois de stage nous ont apporté dans cet environnement professionnel, scientifique et international, qu'est le CERN à Genève.

Toutefois, avant d'entrer dans le vif du sujet de ce mémoire, nous tenons à informer le lecteur qu'une grande partie des figures, servant à illustrer certains points développés tout au long de ce mémoire, disposent de légendes exprimées en anglais.

Chapitre 1:
LE CERN

Pour réaliser cette courte introduction qui explique ce qu'est le CERN (Commission Européenne pour la Recherche Nucléaire), quels types de recherches y sont poursuivis et dans quel but, nous nous sommes aidés d'un fascicule [CERN91] que nous avions reçu lors de notre visite guidée des installations.

1. Le CERN, une organisation internationale

Le CERN, situé à côté de Genève, à cheval sur la frontière franco-suisse, est l'un des plus grands laboratoires du monde pour la recherche en physique des particules. Des installations de taille et d'importance comparables n'existent actuellement qu'aux Etats-Unis et dans l'ancienne Union soviétique.

En 1945, l'Europe ressortait affaiblie et appauvrie des six années de guerre qu'elle venait de vivre. Et comme les moyens manquaient aux états individuels pour financer isolément les instruments de recherche coûteux et indispensables à la science moderne, beaucoup de chercheurs européens émigrèrent aux Etats-Unis où les laboratoires étaient au contraire très modernes et bien équipés.
C'est donc essentiellement en vue de les retenir en Europe qu'un groupe de scientifiques et de politiciens imagina une aventure nouvelle pour la science: un laboratoire scientifique européen.
A l'époque déjà, il était clair qu'un établissement scientifique de pointe avait besoin d'installations d'une taille et d'une complexité dépassant les moyens individuels des nations.

C'est ainsi qu'en 1954, douze états unirent leurs ressources et créèrent le CERN pour rassembler l'essentiel du potentiel européen de recherche, relançant ce qu'on appelait alors, par approximation, la "recherche nucléaire". Actuellement, le CERN comprend 19 états membres européens: l'Allemagne, l'Autriche, la Belgique, le Danemark, l'Espagne, la Finlande, la France, la Grèce, la Hongrie, l'Italie, la Norvège, les Pays-Bas, la Pologne, le Portugal, la République slovaque, la République tchèque, le Royaume-Uni, la Suède et la Suisse.

2. Les recherches menées au CERN

Plus exactement, les recherches menées au CERN ont trait, en réalité, à la physique des particules plutôt qu'à la recherche nucléaire à proprement parler. On étudie ainsi les structures ultimes de la matière, une recherche qui n'a que de très lointains rapports avec la fission nucléaire qui se rapporte, quant à elle, à la production d'énergie à partir des noyaux de certains atomes comme celui de l'uranium.
D'ailleurs, la Convention qui établit le CERN définit très rigoureusement ses tâches et lui interdit formellement toute recherche militaire; le CERN ne doit pas davantage être confondu avec une centrale productrice d'énergie. Le laboratoire du CERN a pour seul but l'étude de la relation entre l'énergie et la matière, et de tout ce que cette relation met en évidence, comme par exemple la structure de la matière dans l'infiniment petit et les forces les plus fondamentales qui agissent dans la nature.

Pour concevoir et construire les machines sophistiquées du CERN et assurer leur bon fonctionnement, pour aider à la préparation, l'exécution, l'analyse et l'interprétation des expériences scientifiques complexes et pour effectuer la multitude de tâches nécessaires dans une organisation aussi grande et aussi spécialisée, le CERN emploie environ trois mille personnes dont les qualifications recouvrent un éventail de compétences et de professions: ingénieurs, physiciens, informaticiens, techniciens, artisans, administrateurs, secrétaires, ouvriers, etc...

Dès son ouverture, le CERN permit peu à peu aux physiciens du vieux continent de rétablir la recherche européenne au niveau d'excellence qu'elle avait connu avant la guerre et de combler ainsi le retard pris sur les Etats-Unis. Le CERN renouait du même coup avec une ancienne tradition européenne d'échanges scientifiques et intellectuels et contribuait, dès ses débuts, à rétablir des contacts amicaux et fructueux entre des pays que la guerre avait dressés les uns contre les autres.

On débuta les travaux par la construction d'un accélérateur de particules: un synchro-cyclotron destiné à accélérer des protons. Il ne s'agissait nullement d'un projet ambitieux, même en comparaison avec les normes modestes de l'époque. Mais il allait permettre de lancer rapidement un programme de recherche et d'acquérir une expérience précieuse.
Notamment, il fallait obtenir des protons seuls dépourvus d'électrons. Pour cela, il suffit de prendre un atome d'hydrogène qui ne contient qu'un seul électron et qu'un seul proton. Ces derniers sont attirés entre eux par une certaine force de "liaison". En appliquant sur le proton une force supérieure à cette force de "liaison", on obtient seulement le proton.

Maintenant que l'on détient tous ces protons, il ne reste plus qu'à leur donner une certaine vitesse grâce à l'accélérateur. La technique utilisée est la suivante: sur une certaine portion de l'accélérateur, on place à la suite les unes des autres des piles, de sorte que la borne positive d'une pile touche la borne négative de la pile suivante, exactement comme dans tout appareil électrique fonctionnant avec des piles. De cette manière, si on présente la borne négative de la première pile de la chaîne au proton, qui est une particule de charge positive, il est attiré vers elle. Au moment où le proton atteint cette première borne négative, on change la polarité des piles de sorte que les bornes négatives des piles deviennent positives et inversement. Le proton est alors maintenant attiré par la borne négative suivante avec une vitesse plus importante.
Et on réitère ce processus jusqu'à ce qu'on atteigne la fin de la chaîne de piles.

En parallèle, le CERN commençait à construire une machine très puissante: le synchotron à protons (le PS). Celui-ci est entré en service en novembre 1959 et, pendant un certain temps, il fut l'accélérateur de particules le plus puissant au monde.
Plus de trente ans plus tard, cette machine reste la cheville ouvrière du système d'accélérateurs interconnectés du CERN.

Toutefois, il a depuis fort longtemps cessé d'alimenter directement en particules les expériences du CERN. La plupart du temps, il fournit aux corpuscules (protons, antiprotons, électrons, positons et noyaux) leur premier grand gain d'énergie avant de les injecter dans des machines plus grandes et plus récentes qui les portent ensuite à des énergies encore plus élevées.

Un peu plus tard, le CERN fit construire un nouvel accélérateur de sept kilomètres de circonférence, le super PS (SPS) qui fut à l'origine de nombreuses découvertes historiques comme celle des particules W et Z, les messagers de la force nucléaire faible.

C'est en juillet 1989 que le dernier accélérateur en date fut opérationnel, le LEP (Large Electrons Positons collider). Il s'agit d'un accélérateur de 27 kilomètres de diamètre situé à une profondeur moyenne de 100 mètres sous le territoire français et le territoire suisse. Des faisceaux d'électrons et des faisceaux de positons (particules inverses des électrons) sont lancés en sens inverse dans l'anneau souterrain à une vitesse proche de celle de la lumière et projetés les uns contre les autres.

Une chaîne de machines interconnectées fournit les faisceaux utilisés dans les expériences du LEP, mais également ailleurs sur le site, et permet de mener un programme de recherche d'une richesse et d'une diversité sans égal.

Le programme de recherche du CERN s'appuie sur une panoplie de grandes machines, sans équivalent dans le monde, et attire des milliers de scientifiques venus des Etats membres et d'ailleurs: Etats-Unis d'Amérique, Canada, Japon, Russie, Chine, Israël, Inde, etc... (Lorsqu'ils travaillent au CERN, ces attachés scientifiques sont rétribués par leur université ou leur institut de recherche)

3. La réussite du CERN: la coopération scientifique

La coopération scientifique instaurée par le CERN à travers quatre expériences situées à des endroits différents sur le LEP est une sorte de miracle: c'est en tout cas le seul exemple de laboratoire international qui fonctionne à cette échelle. Non seulement les chercheurs qui y travaillent viennent de toutes les parties du monde, mais encore les machines elles-mêmes sont le produit des efforts conjugués des industries européennes de pointe: il n'est pas rare qu'un instrument fonctionnant avec la précision de quelques millièmes de millimètre soit composé de pièces de provenances très diverses et qui pourtant s'ajustent impeccablement.

Le CERN est une organisation "transparente" en ce sens que tous les résultats de ses expériences sont publiés et accessibles aux chercheurs du monde entier.
Le but est d'améliorer notre connaissance du monde et de l'univers et de mieux comprendre les lois qui régissent la nature, c'est-à-dire la science pure, sans objectif technologique ou commercial immédiat.
Cependant, même à court terme, cette recherche intensive pousse la technologie moderne aux limites du possible par ses exigences extrêmes de précision et de rapidité de réponse.
Les innovations technologiques que le CERN peut inventer pour la création de telle ou telle de ses machines sont également du domaine public: ces technologies ne sont pas brevetées et chacun peut s'en servir librement. En voici quelques exemples: les scanners pour le diagnostic médical, les marqueurs radioactifs, l'implantation ionique, etc...

Les Etats Membres gouvernent le CERN par l'intermédiaire d'un Conseil siégeant deux fois par an. Un comité de physiciens y débat régulièrement les grandes orientations de la recherche et les propose aux Etats Membres. Une administration efficace exécute la politique décidée par le Conseil et soumet sa gestion financière à un Comité des Finances. Le budget annuel du CERN s'élève actuellement à environ 23 milliards de francs belges, ce qui représente, pour chaque habitant des Etats Membres, environ 100 francs belges par an.
Pour ce prix, somme toute modique, le CERN se maintient avec succès à la pointe de la recherche mondiale en physique fondamentale. Indépendamment de ses résultats, le fait même de la recherche et des talents concentrés au CERN assure à l'Europe un rayonnement scientifique et culturel inestimable.

4. Quelle est l'utilité des recherches menées au CERN ?

Ecrasé par l'immensité, l'homme contemple avec humilité la voûte céleste et s'interroge. Bien que moins évidente, la structure de l'Univers à l'échelle microscopique que révèlent les expériences en physique des particules est tout aussi impressionnante. Le but de la physique est de découvrir les lois de l'Univers, c'est-à-dire de généraliser à partir des connaissances obtenues dans un certain ensemble de conditions expérimentales.
En astrophysique et en cosmologie, on applique au cosmos la science acquise sur terre pour apprendre comment est organisé l'Univers et retracer son évolution. Les cosmologistes considèrent maintenant que l'Univers a pris naissance dans une puissante explosion initiale: le "Big Bang".

Pour découvrir les lois qui régissent notre monde et l'ensemble de l'Univers et essayer de remonter dans le temps afin d'arriver de plus en plus près du moment du "Big Bang" (actuellement, on est remonté jusqu'à 10^{-12} seconde après le "Big Bang"), le CERN sonde les constituants les plus intimes de la matière.
Pour cela, il utilise notamment la plus grande machine scientifique du monde, le fleuron des instruments de recherche du CERN: le LEP.
Des milliers de grands aimants maintiennent les faisceaux d'électrons et de positons sur leurs orbites précises de sens opposés dans un tube en alliage d'aluminium constituant le plus long chemin sous vide dans le monde. A côté de cela, les puissants équipements de radiofréquence portent les faisceaux à de très hautes énergies.

Pour observer ce qui se passe quand des particules interagissent, les physiciens ont besoin de "détecteurs" dans lesquels des millions de particules entrent en collision à chaque seconde, détecteurs d'autant plus complexes que chacune de ces collisions produit une "explosion" pouvant compter plusieurs centaines de fragments.
Il existe quatre puissants détecteurs (L3, OPAL, DELPHI et ALEPH), tous de la taille d'un immeuble de quatre étages et possédant des ordinateurs très perfectionnés et très rapides. Ils sont disposés sur l'anneau du LEP pour arrêter, enregistrer et analyser les fragments de matière émergeant des collisions. Il faut savoir que l'information recueillie dans une seule collision suffirait à remplir un annuaire téléphonique, de sorte que les données doivent être très rapidement enregistrées.
Pour la plus grande partie d'entre elles, elles relèvent d'une physique bien connue et les physiciens les éliminent au "déclenchement"; pour cela, ils pré-établissent des critères garantissant que seules les données intéressantes seront retenues en vue d'une analyse. Afin que le détecteur soit prêt pour la collision suivante, ces déclenchements sélectifs doivent agir en quelques micro-secondes.

Les résultats de ces recherches sont difficiles à traduire en langage courant. Mais il est bien connu que nombre de découvertes importantes ont été faites au CERN au cours des vingt dernières années, et qu'elles sont dues dans une large mesure aux machines très sophistiquées construites sur place par des ingénieurs de grande compétence. Cet alliage de science et de technologie a par exemple permis dans le milieu des années 80, de créer, de conserver et d'accélérer des antiprotons pour des expériences d'un type nouveau: des collisions "matière contre anti-matière".

5. La nécessité d'un réseau informatique bien développé et bien géré

Au CERN, tous les employés travaillent, d'une manière plus ou moins éloignée, pour la recherche en physique nucléaire. A part les physiciens qui s'y consacrent directement, toutes les autres personnes sont présentes afin de les aider et de faciliter leur travail de recherches, c'est-à-dire de leur donner un "soutien logistique".

Les physiciens réalisent leurs expériences en physique des particules à l'aide de machines hautement sophistiquées et d'ordinateurs possédant une grande puissance de calcul qui analysent et

traitent des quantités très importantes de données endéans des temps très courts (quelques micro-secondes).
Pour récolter et transférer ces prodigieuses quantités de données, le CERN a dû développer des réseaux interconnectant ces machines et ces ordinateurs.

Au départ, ces différents réseaux étaient seulement localisés dans des bâtiments ou des ensembles de bâtiments et n'étaient pas interconnectés entre eux.
Mais peu à peu, on a commencé à relier ces réseaux d'abord entre eux, puis ensuite avec le monde extérieur, pour deux raisons principales. D'abord, pour faciliter le transit des informations au sein de l'organisation. Ensuite pour communiquer avec l'extérieur et permettre la collaboration entre chercheurs et centres de recherche se trouvant éparpillés un peu partout dans le monde ainsi que pour offrir l'opportunité, à quiconque y serait intéressé, de suivre l'évolution des recherches scientifiques (par exemple grâce au World Wide Web dont le CERN est à l'origine) car n'oublions pas que le CERN est une organisation qui se veut "transparente".

6. Une brève description de la topologie du réseau du CERN

Le CERN est un précurseur en matière de réseaux. Il dispose de nombreux sous-réseaux de types différents (Ethernet, FDDI...) interconnectés entre eux par l'entremise d'un "back-bone" en fibre optique. Ce "back-bone" est composé de seize réseaux de type FDDI qui sont interconnectés par un "super bridge", le Giga-Switch, possédant une très grande capacité de traitement. De cette manière, on peut, à partir d'un quelconque ordinateur, atteindre ou se connecter sur n'importe quelle machine au CERN, à condition qu'on en ait l'autorisation (un "login" et un mot de passe).
A chacun de ces seize réseaux de type FDDI sont reliés de nombreux sous-réseaux, tels que des LANs (Local Area Network) qui peuvent encore être décomposés en segments pour pouvoir isoler les flux de données.
A la figure 1.1., nous trouvons une illustration qui représente le "back-bone" du CERN avec son "Giga-Switch et ses seize réseaux de type FDDI.

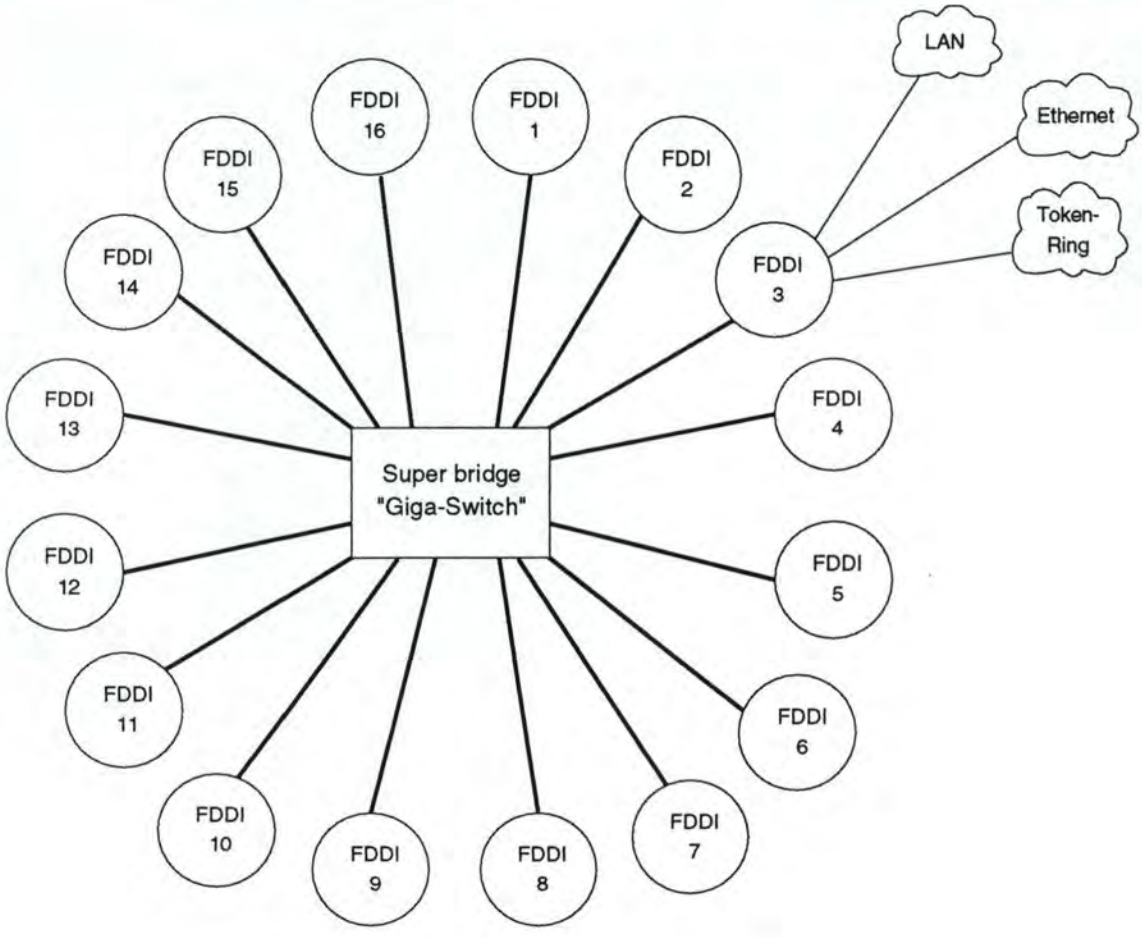


Figure 1.1. Le "back-bone" du CERN.

Ces différents sous-réseaux sont interconnectés grâce à des bridges et servent à relier entre eux des terminaux, des stations de travail ou des ordinateurs. Ces bridges arrivent à discerner si une information envoyée sur le réseau à partir d'une machine est destinée à une autre machine se trouvant sur le même réseau local ou sur un autre réseau. Dans le second cas, ils feront transiter l'information vers le bon réseau. Un des avantages des bridges est d'éviter de surcharger l'ensemble du réseau avec des informations inutiles.
Sur le réseau de CERN viennent également se greffer des routeurs grâce auxquels l'organisation internationale communique avec ses différents partenaires et le monde extérieur. L'ensemble de ces routeurs constitue le réseau externe du CERN.
La topologie complète du réseau du CERN est représentée en annexe 1 située à la fin de ce mémoire.

Mais un réseau d'une telle taille requiert un grand souci de gestion et il y a quelques années, il n'existait pas encore sur le marché de produits qui répondent aux besoins spécifiques du CERN. C'est pour cette raison que l'organisation internationale avait décidé, à l'époque, de développer elle-même un logiciel, appelé CAW, pour gérer l'entièreté de son réseau.

Lors de ces dernières années, le responsable du groupe CS ("Communications Systems") de la division CN ("Computers & Networks") s'est rendu compte que la maintenance et le développement du logiciel CAW coûtaient très cher en personnel et en temps de travail. Il a donc pris la décision, avec le responsable de la division CN, d'acquérir un logiciel commercial pour la gestion du réseau du CERN.
Son choix s'est porté sur le logiciel SPECTRUM, créé par une firme américaine, Cabletron System. Ce logiciel a été conçu pour permettre la gestion d'un réseau de communication à grande échelle. Ses grandes caractéristiques et ses principales fonctionnalités seront décrites dans le chapitre suivant.

Chapitre 2:
Le logiciel SPECTRUM

Dans ce chapitre, nous allons présenter le logiciel de gestion de réseaux SPECTRUM sur lequel nous avons travaillé tout au long de notre stage au CERN. Nous développerons ses principales fonctionnalités et caractéristiques, nous donnerons ses atouts qui lui ont permis d'être choisi parmi tous les logiciels de gestion de réseaux présents sur le marché et nous verrons dans quelle mesure SPECTRUM peut être considéré comme un système intégré.
Mais avant d'étudier SPECTRUM en détail, nous allons introduire deux notions importantes. Il s'agit d'abord de la notion de "réseau" par laquelle nous expliquerons ce que sont les réseaux informatiques, pourquoi ils ont été développés et comment ils sont interconnectés. Ensuite, nous verrons la notion de "gestion de réseaux" qui constitue un point fondamental dans le domaine actuel des réseaux informatiques. Nous mentionnerons quelques caractéristiques importantes dont devrait disposer tout bon logiciel de gestion de réseaux.
Pour cette introduction, nous nous sommes principalement inspirés du livre de M. Rose [ROSE91].

1. Notion de "réseau"

Un réseau de télécommunication constitue un moyen par lequel des entités se communiquent des informations. Ces entités peuvent être soit des personnes, soit des machines.
Dans le cadre de ce mémoire, nous nous intéresserons essentiellement aux réseaux informatiques permettant à des ordinateurs autonomes ou manipulés par des hommes de se communiquer de l'information.

Au début de l'ère de la communication entre ordinateurs dans les années 60, il s'agissait simplement de transférer d'une manière fiable des bits d'un gros ordinateur central vers des terminaux et inversement.
Avec l'évolution de la technologie dans les années 70, les terminaux sont devenus des stations de travail: celles-ci sont connectées à un simple réseau qui transporte des données sous la forme de paquets et au moyen duquel elles peuvent communiquer entre elles.
Ce n'est qu'à partir du milieu des années 80 que divers facteurs économiques et technologiques ont permis l'interconnexion de ces simples réseaux.

Dans le réseau Internet, plusieurs réseaux sont connectés ensemble à l'aide de passerelles et d'un protocole d'interconnexion. Les passerelles, souvent appelées "des routeurs", utilisent ce protocole, sans se préoccuper des détails sous-jacents des sous-réseaux, ceci dans le but de fournir un service uniforme à travers l'ensemble des sous-réseaux.

Par exemple, un réseau se trouvant dans un site pourrait consister en un LAN utilisant une technologie Ethernet. Ce réseau, ainsi que d'autres LANs situés non loin de lui, pourraient être interconnectés par un réseau régional qui serait composé de plusieurs routeurs avec des connexions point-à-point reliant chacun de ces LANs.
De même, ce réseau régional peut être couplé avec d'autres réseaux régionaux pour former un "back-bone", c'est-à-dire un réseau au niveau national par le biais d'un nouvel ensemble de routeurs.
Finalement, on peut encore connecter ce réseau national à d'autres réseaux nationaux pour disposer de connexions internationales.

On a vu qu'un réseau régional considère les LANs qu'il relie comme de simples réseaux logiques.
Si l'on décide de changer de direction et de prendre celle du "top-down", un LAN pourrait être réellement composé de plusieurs segments (partie d'un réseau) interconnectés entre eux par des bridges, des répétiteurs ou d'autres routeurs.
Cette division en segments présente l'avantage de pouvoir isoler au sein d'un réseau comme un LAN les différents flux de données qui y transitent.

Bien que chaque sous-réseau puisse être basé sur des technologies sous-jacentes totalement différentes comme Ethernet, Token-Ring ou X-25, chacun possède ses propres règles de transmission de sorte que tous les hôtes reliés à un même sous-réseau disposent d'une vue commune de ce sous-réseau.
C'est ce qu'on appelle la puissance d'abstraction de l'interconnexion des réseaux. Par l'utilisation d'un protocole commun et d'algorithmes, même le réseau le plus hétéroclite et constitué d'une myriade de technologies peut être considéré comme une simple connexion point-à-point sur un réseau physique homogène.

2. Notion de "gestion de réseaux"

Depuis plusieurs années, on peut remarquer un accroissement de la taille et de la complexité des réseaux informatiques. On doit attribuer cela en grande partie à l'explosion du marché des ordinateurs et des composants de réseaux comme les stations de travail, les bridges, les routeurs, les nouvelles applications de transfert d'information, les supports pour transmettre les données, etc...
Mais cette explosion du marché a pour conséquence l'apparition d'une multitude de fabricants fournissant des composants de réseau très hétérogènes.
Pour interconnecter tous ces différents composants, il est donc indispensable qu'ils implémentent au moins tous un même protocole de transport et d'interconnexion, comme TCP/IP.
Il est clair qu'un système de gestion de réseaux spécifique à un fabricant particulier n'est plus imaginable dans la situation actuelle.
Il convient donc de disposer d'un système "ouvert" pour gérer tous ces réseaux de types différents qui sont interconnectés.

Dans le cadre spécifique des réseaux, nous assistons également à une évolution des techniques et du matériel utilisés qui conduit à la mise au point de mécanismes de gestion de réseaux toujours plus exigeants.

Pour disposer d'un réseau fournissant un bon rendement, des outils efficaces d'aide à la gestion de réseaux sont devenus indispensables. [NAC95]

Ces outils doivent être capables:

- d'aider les administrateurs du réseau à prédire son évolution;
- de gérer des réseaux de grande taille ainsi que leur interconnexion avec d'autres réseaux;
- de détecter les erreurs provoquées par le changement de matériel ou l'ajout d'équipements nouveaux;
- de fournir une technique de gestion uniforme dans un environnement d'interconnexion dans lequel coexistent divers équipements et divers produits. A travers cette technique, il convient de gérer aussi bien un sous-réseau particulier que l'ensemble du réseau Internet.

Dans le contexte de la gestion de réseaux, l'ISO a défini, dans le cadre de son Modèle de Référence OSI, cinq domaines qui devraient être couverts par un système de gestion de réseaux [NAC95]:

1. la gestion des erreurs, qui s'occupe de détecter et de corriger les problèmes qui surviennent sur le réseau;

2. la gestion des configurations, qui prend en charge la localisation des éléments du réseau (leur emplacement, leur nom, leur type...) ainsi que la gestion logique et physique du réseau;

3. la gestion des performances, qui donne des statistiques sur les équipements du réseau (le taux d'erreur, le nombre de bytes par seconde...). En fonction de ces statistiques, l'administrateur du réseau décidera des actions à entreprendre;

4. la gestion financière, qui évalue les coûts du réseau en vue de les facturer aux différents utilisateurs;

5. la gestion de la sécurité, pour éviter que n'importe quel utilisateur ne puisse accéder à toutes les informations du réseau (utilisation d'un mot de passe identifiant l'utilisateur et lui donnant accès à seulement certaines informations);

Et pour terminer, la tâche de gestion d'un réseau peut être divisée en deux classes distinctes selon [HEY91]. La première est la gestion au jour le jour qui consiste à rassembler des informations sur les appareils constituant le réseau (bridge, routeur...), à réaliser la maintenance, à résoudre les pannes... Cette première tâche est effectuée par des personnes qui sont responsables du bon fonctionnement du réseau.
La deuxième tâche consiste, quant à elle, aux opérations de gestion. Les opérateurs doivent disposer constamment d'une information sur l'état du réseau et, en cas de problème, être à même de savoir comment y faire face.

3. SPECTRUM: un logiciel pour la gestion de réseaux

Pour décrire le logiciel SPECTRUM, nous nous sommes basés sur les guides et modes d'emploi fournis par la société Cabletron. Dans la bibliographie située à la fin de ce mémoire, ils correspondent aux références [SPEC1], [SPEC2], [SPEC3], [SPEC4], [SPEC5] et [SPEC6]. Nous avons également utilisé un document trouvé sur Internet, [INT94].

Le logiciel SPECTRUM est un logiciel de gestion de réseaux extensible et intelligent conçu par la société américaine Cabletron pour gérer un réseau de communication à grande échelle.

Le logiciel dispose de plusieurs caractéristiques que nous développerons ultérieurement dans ce chapitre. Mais avant d'entrer dans les détails de la description du logiciel, nous allons définir et expliquer certains concepts importants.

3.1. La signification des termes

Pour éviter tout risque d'ambiguïté, nous allons tout d'abord donner une explication de certains termes utilisés dans la suite de ce travail qui pourraient porter à confusion. Les explications de ces termes ne constituent pas des définitions strictes. En effet, dans d'autres littératures, certains termes pourraient avoir des significations légèrement différentes de celles présentées dans la suite de ce mémoire.

3.1.1. Un "pingable"

Un "pingable" est un équipement du réseau chargé de vérifier si les connexions avec certaines machines externes au réseau du CERN sont toujours établies. En réalité, il s'agit d'un routeur qui, en plus de sa fonction de routage, envoie via le réseau des paquets "ping" vers une machine externe pour s'assurer que la connexion avec celle-ci est toujours bien établie. A ce paquet "ping", la machine doit répondre au "pingable" en renvoyant un paquet "pong".
De cette manière, lorsqu'une machine externe émet un paquet "pong" en réponse à un paquet "ping" en provenance d'un "pingable", ce dernier peut considérer que la connexion est toujours établie.

Bien que, comme nous venons de le dire, un "pingable" peut être considéré comme un routeur "spécial", le CERN réalise toujours une distinction entre ces deux types d'équipement. Dans la suite de ce mémoire, nous continuerons donc à discerner les "pingables" et les routeurs.

3.1.2. Un agent

Un agent est une entité constituant un élément du réseau de télécommunication et supportant un protocole de gestion de réseaux: un bridge, un routeur, un "pingable", une station de travail, un PC...

3.1.3. Un système de gestion de réseaux

Un système de gestion de réseaux est un système qui tourne sur une machine située au sein d'un réseau et qui se charge de gérer des composants de ce réseau ainsi que le réseau lui-même. Pour réaliser cette tâche, il analyse le trafic qui transite sur le réseau et effectue fréquemment un contrôle sur certaines valeurs caractérisant l'état d'un composant du réseau.
Ainsi, il informe les opérateurs et administrateurs sur tout problème survenant sur le réseau pour que ceux-ci puissent y remédier.

3.1.4. Une interface

Un bridge (ou un routeur) contient toujours au moins deux interfaces. Chacune de ces interfaces correspond à une connexion physique vers un autre composant du réseau tel qu'un sous-réseau, un routeur, un autre bridge...

Dans le cadre de ce mémoire, nous avons considéré une interface selon deux aspects différents pour faciliter la compréhension de certains concepts qui seront introduits dans le chapitre 4 consacré à l'explication de notre premier projet.
En effet, dans SPECTRUM, une même interface est caractérisée par deux identifiants. Le premier qui est vu lorsqu'on se place du côté du bridge et que nous appellerons l'identifiant de l'interface "*intérieure*". Le deuxième qui est vu lorsqu'on se situe du côté du composant connecté au bridge et que nous appellerons l'identifiant de l'interface "*extérieure*".
Le figure 2.1. illustre les deux manières par lesquelles une interface peut être vue.

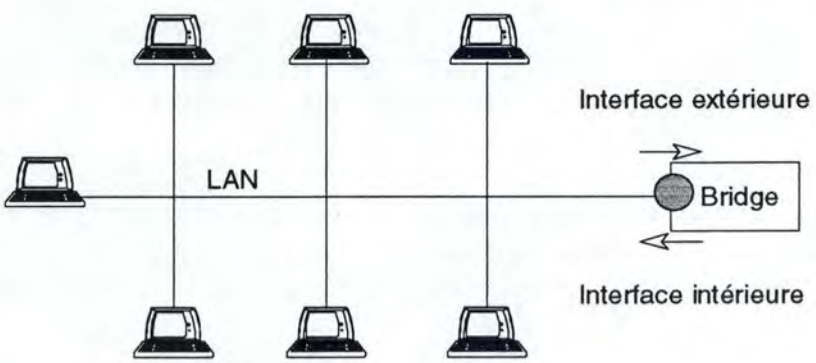


Figure 2.1. Différence entre interface "*intérieure*" et interface "*extérieure*".

3.1.5. Une vue

Une vue est une fenêtre appartenant au logiciel SPECTRUM qui correspond à une représentation visuelle d'une partie du réseau ou d'un ensemble d'informations sur un ou plusieurs équipements (bridges, routeurs, interfaces...) du réseau.

3.1.6. Le SpectroSERVER

Le SpectroSERVER peut être considéré comme le système de gestion de réseaux du logiciel SPECTRUM et en constitue son "centre nerveux" dans le sens où il lui fournit une certaine "intelligence" et contient les modèles des composants du réseau, de l'information sur ces modèles ainsi que toutes leurs interactions. Il s'agit du programme serveur qui s'occupe de récolter, de gérer et de contrôler les différents attributs des agents qu'il a sous sa responsabilité. Il veille

également à un bon niveau de sécurité et offre certaines facilités aux utilisateurs. Il a en charge la gestion de la base de connaissances de SPECTRUM qui comprend toutes les informations sur l'ensemble des équipements du réseau ainsi que tous les événements qui se sont produits sur le réseau depuis le lancement du SpectroSERVER. Son principal composant est la "Virtual Network Machine" (VNM) qui représente une modélisation de la machine sur laquelle tourne le SpectroSERVER.

3.1.7. Le SpectroGRAPH

Le SpectroGRAPH est le programme client qui constitue une interface graphique orientée-icônes avec laquelle l'utilisateur peut interagir. Il permet de voir les différentes représentations visuelles du réseau contenues dans la base de connaissances du SpectroSERVER. Le SpectroGRAPH est utilisé pour gérer une grande variété de composants du réseau et est également considéré comme un écran de contrôle virtuel du réseau au moyen duquel l'utilisateur peut envoyer des commandes pour contrôler un quelconque composant du réseau.
Lorsqu'un utilisateur lance, localement ou non, un SpectroGRAPH, il est nécessaire que le SpectroSERVER tourne afin que le SpectroGRAPH puisse s'y connecter.
Plusieurs SpectroGRAPH peuvent être lancés en même temps à partir de stations de travail distinctes.

La figure 2.2. illustre la manière dont dépendent le SpectroGRAPH et le SpectroSERVER l'un de l'autre.

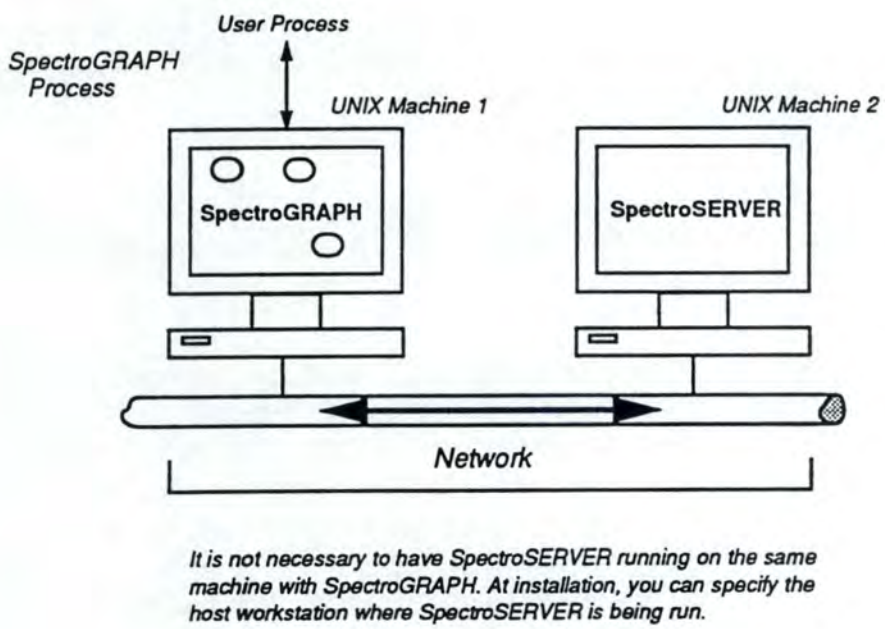


Figure 2.2. Dépendance entre le SpectroGRAPH et le SpectroSERVER.

3.1.8. Un segment

Un segment représente une partie du support électrique formant un (sous-)réseau sur lequel sont connectées des machines ou des stations de travail. Ces segments peuvent être reliés entre eux au moyen de routeurs ou de bridges.
Au CERN, on parle d'un segment logique pour un LAN et d'un segment physique pour le tronçon du support à proprement parler.

3.1.9. Un modèle

Un modèle correspond à une représentation d'un équipement ou d'un groupe d'équipements physiques du réseau (bridge, routeur, câble, port...). Cette représentation sous la forme d'une icône permet à l'utilisateur de surveiller et de contrôler l'équipement par l'entremise du SpectroGRAPH. Tout modèle se trouve dans la base de connaissances de SPECTRUM et possède plusieurs caractéristiques telles que son nom, son type, son adresse physique, l'intervalle de temps séparant deux "pollings" successifs, une brève description de l'appareil, etc...

3.1.10. Une trappe

Une trappe constitue un moyen par lequel un agent du réseau peut avertir le système de gestion de réseaux, dans notre cas le SpectroSERVER, de la survenance d'un événement extraordinaire et inattendu qui pourrait avoir des conséquences au sein du réseau. Il s'agit en réalité d'un message que l'agent envoie sur le réseau à destination du système de gestion et contenant des informations sur l'événement qui vient de se produire. Le format du message est défini d'une manière telle qu'il vise à minimiser le nombre d'informations transmises pour décrire l'événement, ceci afin de ne pas trop surcharger le réseau.

3.2. La base de données contenant toutes les informations nécessaires à la gestion d'un réseau

La base de données qui reprend l'ensemble des informations nécessaires à la gestion d'un réseau est mieux connue sous le nom de MIB, "Management Information Base".
Cette MIB est organisée selon une structure hiérarchique, généralement présentée sous la forme d'une arborescence. En réalité, l'arborescence de la MIB constitue déjà elle-même une sous-arborescence de l'arbre décrivant l'entièreté des objets appartenant au monde des réseaux (produits, documentations, protocoles, standards...).
A chaque noeud de l'arborescence de la MIB correspond un type d'objet, et à chaque feuille correspond une variable caractérisant partiellement l'agent.

Tout agent qui doit être géré sur un réseau dispose, pour chacune de ses interfaces, d'une base de données d'informations nécessaires à la gestion. Cette base de données est une instanciation d'une partie (d'un sous-arbre) de la MIB, de manière telle que des agents de même type implémentent la même partie de la MIB. Le système de gestion possède, pour chaque agent, une vue virtuelle de l'instanciation de la MIB de l'agent. L'ensemble de ces vues virtuelles définit l'information disponible pour le système de gestion.

3.2.1. Caractéristiques d'un type d'objet

Chaque type d'objet décrit dans la MIB est caractérisé par trois aspects:

- un nom

Il est assigné par une autorité administrative sous la forme d'une suite d'entiers qui détermine de manière univoque sa place dans l'arborescence.

- une syntaxe abstraite

Elle définit la structure de donnée abstraite du type d'objet. Cette structure peut désigner un "OCTET STRING", un "INTEGER" ou toute autre construction plus complexe appartenant à la notation ASN.1.

- un (type d')encodage

Il définit la manière dont les instances des types d'objet sont représentées relativement à la syntaxe de l'objet.
Il définit également la manière dont les différentes instances sont codées pour être transmises sur le réseau.

Actuellement, la MIB distingue plusieurs groupes d'objets. En voici quelques-uns:

- groupe système décrivant le système de l'agent géré;
- groupe interfaces donnant toutes les connexions réseau de l'agent;
- groupe tcp caractérisant le protocole de contrôle de transmission de l'agent;
- groupe ip présentant une description du protocole d'interconnexion.

A chaque groupe d'objets appartiennent un ensemble de variables caractérisant l'agent: son nom, sa date de mise en service, le temps écoulé depuis sa mise en service, la description de son système, les différents attributs pour sa gestion (le nombre de paquets erronés entrant ou sortant par seconde, le nombre de bytes par seconde, le nombre de paquets émis en "multicast" ou en "broadcast",...), etc...

3.2.2. Principe d'identification d'un type d'objet

Chaque type d'objet possède un nom qui l'identifie de manière univoque au sein de la MIB. Cet identifiant (de type "OBJECT IDENTIFIER") est défini selon un schéma bien établi par la notation ASN.1.
Les différents types d'objet sont organisés, sur base de leur identifiant, d'une manière hiérarchique, selon une structure arborescente dans laquelle chaque noeud correspond à un objet. Un noeud est alors caractérisé par un entier qui l'identifie de façon univoque parmi tous les noeuds-fils du noeud dont il dépend directement.

Par conséquent, pour identifier un noeud quelconque dans l'arborescence de la MIB, il suffit de donner la séquence d'entiers qui identifient chacun un noeud dans l'arborescence par lequel on est obligé de passer pour arriver au noeud recherché.

La définition de la MIB est établie de telle manière qu'elle reprend tous les objets nécessaires à la gestion de réseaux. Les types d'objet sont groupés au sein de la MIB en fonction des différents protocoles par lesquels ils peuvent être supportés. Cette répartition permet entre autres de n'implémenter dans la MIB d'un agent à gérer que les objets des groupes pour lesquels l'agent supporte également le protocole.

Chaque agent devant être géré possède une instanciation d'une sous-arborescence de la MIB dans laquelle chaque noeud correspond à une instance du type d'objet défini dans le standard RFC 1213. Bien que ce ne soit pas tout-à-fait correct d'un point de vue sémantique, nous appellerons par convention cette instanciation "la MIB d'un agent".

3.3. Le mécanisme de trappes ou le mécanisme de "pollings"

Pour avoir une bonne gestion du réseau, il est utile que le système de gestion puisse constamment surveiller le bon fonctionnement de l'ensemble des composants du réseau.
Il doit essayer de "garder un oeil" sur chaque agent en vérifiant des valeurs clés telles que le nombre de bytes qui transitent par seconde, le nombre de paquets erronés par seconde, le nombre de paquets émis en broadcast, le nombre de paquets émis en multicast...
C'est en fonction de ces différentes valeurs que le système de gestion peut constater si un agent présente, oui ou non, un problème.

Cette surveillance peut s'effectuer de deux manières différentes:

- le mécanisme de "polling" qui constitue la technique principale utilisée dans la communication entre un système de gestion et un agent. C'est au système de gestion qu'il revient d'aller interroger les agents à intervalles de temps réguliers;

- le mécanisme de trappe qui constitue le moyen pour un agent d'avertir le système de gestion de la survenance d'un événement inattendu qui pourrait avoir des conséquences importantes au sein du réseau.

Depuis la naissance des systèmes de gestion, on entretient un débat sans fin dans le but de savoir laquelle de ces deux approches choisir pour détecter les alarmes qui pourraient survenir à partir d'un composant du réseau.
Nous allons tout d'abord donner les avantages et les inconvénients de chacune de ces deux approches.
Ensuite, nous montrerons que l'on peut arriver à un compromis entre ces deux approches pour concilier leurs avantages tout en restreignant leurs inconvénients.

3.3.1. Le système de trappes

Quand un événement exceptionnel se produit avec le système de trappes, tel qu'une liaison coupée, l'appareil connecté à cette liaison envoie une trappe au système de gestion afin de l'en informer (pour autant que la liaison coupée ne soit pas celle qui relie l'appareil au système de gestion).

Cette approche est avantageuse dans le sens où le système de gestion est directement mis au courant du problème. Par contre, elle présente plusieurs désavantages:

- elle nécessite des ressources présentes dans l'agent pour générer la trappe;

- si la trappe doit contenir beaucoup d'informations, l'appareil qui est chargé de l'envoyer pourrait passer trop de temps à la fabriquer et pas assez de temps sur des opérations plus utiles. Par exemple, si les trappes nécessitent l'envoi d'un accusé de réception de la part du système de gestion, l'appareil l'attendra et, tant qu'il ne l'aura pas reçu, ré-émettra la trappe, à moins qu'on ne dispose d'un système de trappes idempotent, c'est-à-dire ne demandant pas d'accusé de réception;

- si plusieurs événements exceptionnels se produisent en même temps, une grande partie de la largeur de bande du réseau contiendra des paquets en rapport avec des trappes. Or, l'envoi de ces trappes est indispensable surtout quand il s'agit d'informations indiquant, par exemple, qu'un agent arrive à sa capacité de traitement maximale, ce qui pourrait entraîner à plus ou moins court terme une congestion du réseau si rien n'est entrepris pour y remédier rapidement. Ainsi, si on veut affiner le système de trappes, un appareil pourrait utiliser des "seuils" par événement en dessous desquels il ne génère pas de trappes. Malheureusement, ceci signifierait que l'appareil devrait passer un temps non négligeable à déterminer si, oui ou non, il doit générer une trappe.

Par conséquent, l'utilisation du système de trappes occasionne un impact important sur la performance des agents du réseau, ou du réseau lui-même, ou des deux.
De plus, le système de gestion dispose seulement d'une vue très limitée sur le réseau qui dépend de la bonne utilisation du système de trappes par les agents gérés.
Dans cette optique, on peut considérer le système de gestion comme un système passif attendant qu'on lui rapporte les événements se déroulant sur le réseau.

3.3.2. Le système de "polling"

Avec le système de "polling", le système de gestion interroge périodiquement les agents gérés pour s'informer si "tout va bien". A intervalles de temps réguliers (par exemple toutes les cinq minutes comme c'était le cas pour SPECTRUM), le système de gestion du réseau va aller chercher dans la MIB de chaque agent les informations voulues.

Dans ce cas-ci, le système de gestion (et son administrateur) est seul responsable de la collecte d'informations nécessaires à la gestion du réseau. C'est à lui de déterminer quels sont les agents à consulter, à quel moment interroger un agent, à quelle fréquence, quelles sont les informations qui lui sont utiles ou qui pourraient lui servir...

L'avantage de cette approche se situe dans le fait que c'est le système de gestion qui détient le contrôle pour déterminer l'état du réseau. Il joue un rôle très actif, tandis que chaque agent joue un rôle passif. En effet, l'agent ne fait que fournir les informations sur demande et ne surcharge donc pas le réseau avec divers paquets contenant des informations redondantes ou inutiles à destination du système de gestion.

Le désavantage de cette méthode consiste en un problème de "timing": comment le système de gestion peut-il savoir quels sont les agents qui doivent faire l'objet d'un "polling" et quel est l'intervalle de temps qui doit séparer deux "pollings" consécutifs sur le même agent?
Si cet intervalle est trop petit, la largeur de bande du réseau est gaspillée par le transfert d'informations redondantes; s'il est trop grand, la réponse aux événements catastrophiques est trop lente.

3.3.3. Approche combinée "polling orienté trappes"

Pour essayer de concilier les avantages des deux approches et limiter leurs inconvénients respectifs, le standard Internet pour les systèmes de gestion de réseaux prône une approche combinée "polling orientée trappes".
Ainsi, lors d'une gestion "polling orienté trappes", le système de gestion effectue toujours régulièrement des "pollings" pour suivre l'évolution de certaines valeurs se trouvant dans la MIB d'un agent telles que le nombres de paquets erronés par seconde ou encore le nombre de paquets émis en multicast ou en broadcast.

Et lorsqu'un événement extraordinaire se passe, l'appareil géré génère une simple trappe au système de gestion sans demande d'accusé de réception. Il revient à ce dernier d'aller interroger l'appareil en question pour prendre connaissance de la nature et de l'étendue de son problème.

Ce compromis est assez efficace: l'impact sur les appareils gérés est minimisé, et leurs problèmes peuvent être traités endéans un délai acceptable.
Bien sûr, puisque les trappes sont envoyées au système de gestion sans demande d'accusé de réception, il se peut que la trappe n'arrive jamais à bon port. C'est pour cette raison qu'on aura toujours besoin d'un système de "polling" à intervalles de temps pas trop grands comme sécurité.

3.4. Le protocole SNMP: "Simple Network Management Protocol"

Le protocole de gestion de réseaux SNMP est un protocole de seconde génération. Il a très vite rencontré un succès sans précédent dans son développement parce qu'on avait un besoin urgent d'un protocole simple pour gérer l'ensemble du réseau Internet basé sur TCP/IP. [ROSE91]
Bien qu'on ne puisse pas gérer tous les agents du réseau Internet, certains ne supportant pas encore ce protocole de gestion, SNMP est implémenté sur une large gamme de plate-formes hardware et software dans de nombreuses lignes de produits des vendeurs.

SNMP est un protocole décrivant la communication entre un système de gestion de réseaux et les différents composants du réseau qu'il doit gérer, les agents.
Comme il se veut simple, il offre seulement les fonctionnalités de base nécessaires aux applications de gestion.

Comme nous avons pu le constater au début de ce chapitre, chaque agent est caractérisé par des valeurs qui sont situées dans sa MIB. Pour contrôler un agent, le système de gestion doit être en mesure de vérifier la valeur de ces différentes variables et d'en modifier quelques-unes.
De plus, nous avons également vu qu'un agent devait avoir la possibilité d'avertir le système de gestion de la survenance d'un événement extraordinaire.
Sur base de ces quelques fonctionnalités, on peut élaborer des applications de gestion de réseaux assez complexes.

Voici les cinq primitives offertes par le protocole SNMP pour réaliser les quatre types d'opérations :

- GET-REQUEST: Consulter la valeur d'une variable de la MIB d'un agent;
- GET-NEXT-REQUEST: Consulter la valeur d'une variable de la MIB d'un agent qui suit une variable donnée dans l'arborescence;
- SET-REQUEST: Mettre à jour la valeur d'une variable de la MIB d'un agent;
- TRAP: Notifier la survenance d'un événement extraordinaire au sein d'un agent;

- GET-RESPONSE: Fournir une réponse suite à une demande de lecture ou de modification d'une variable de la MIB d'un agent.

Le protocole de gestion de réseaux SNMP utilise le protocole de transport UDP (User Datagram Protocol) qui n'assure pas une communication fiable de bout à bout.
C'est pour cette raison que le protocole SNMP se base sur un mode de communication en deux temps. Cela signifie que, si le système de gestion a déjà envoyé une demande d'information à l'un de ses agents, il peut exécuter une autre tâche avant de recevoir la réponse.

3.4.1. La description de l'architecture du protocole SNMP

Avec le protocole SNMP, le système de gestion de réseaux communique au travers d'un réseau avec les agents dont il a le contrôle à l'aide du protocole de transport sans connexion UDP.
Le système de gestion et l'agent sont tous deux constitués d'une entité SNMP supportant le protocole. Cette entité offre les fonctionnalités de base nécessaires pour la gestion de réseaux à une entité d'application. D'un côté, l'entité d'application de l'équipement qui est chargé de gérer le réseau, c'est-à-dire le système de gestion, effectue les demandes de consultations et de mises à jour des variables de la MIB des différents agents du réseau qu'il gère.
De l'autre côté, les entités d'application des autres agents du réseau renvoient les informations demandées, effectuent les modifications demandées et signalent la survenance de tout événement extraordinaire par le mécanisme de trappes.

3.4.2. Le travail administratif du protocole SNMP

Ce travail consiste essentiellement à déterminer les politiques d'authentification des messages et d'autorisation utilisées entre entités d'application SNMP.
Le protocole de gestion de réseaux SNMP définit la notion de communauté ("community") comme étant une relation entre un agent SNMP et un ou plusieurs systèmes de gestion supportant le protocole SNMP. La sémantique d'une communauté est complexe, bien que la syntaxe reste simple. Une communauté est caractérisée par un nom ("community name") constitué d'une suite d'octets. Chaque octet peut prendre une valeur comprise entre 0 et 255, bien que dans la plupart des cas, les noms de communauté soient constitués exclusivement de caractères ASCII affichables.

Lorsque des messages SNMP sont échangés, ils contiennent deux éléments:

- un nom de communauté qui identifie l'auteur du message;
- une opération SNMP avec ses opérandes associées.

La notion d'authentification de la communauté

Jusqu'à maintenant, seuls les mécanismes triviaux d'authentification sont disponibles. Cela signifie que le nom de la communauté est transmis "en clair" au sein du message SNMP. Si le nom de la communauté correspond à un nom de communauté connu par l'entité SNMP réceptrice, l'entité émettrice est reconnue comme appartenant à la communauté et pouvant traiter avec l'entité réceptrice.

La notion d'autorisation

Cette notion vient ajouter le concept de mode d'accès à une variable de la MIB d'un agent. Ce mode d'accès, défini pour tout objet de la MIB, peut prendre l'une de ces quatre valeurs:

- Read-only
- Read-write
- Write-only
- Not-accessible

Après avoir authentifié le message SNMP, l'agent vérifie que les requêtes formulées respectent bien le mode d'accès défini pour chacune des variables de la MIB.

La notion de délégation ("proxy")

Jusqu'à présent, on a parlé seulement des agents qui pouvaient directement supporter le protocole de gestion de réseaux.
Mais certains équipements du réseau tels que des bridges, des routeurs, des répétiteurs peuvent ne pas supporter le protocole SNMP. Ces éléments sont appelés éléments étrangers ("foreign device").

Pour essayer de disposer d'un système de gestion de réseaux universel qui convienne à tous les composants du réseau, on fait appel au mécanisme de délégation utilisé par des "proxy-agents".
Un "proxy-agent" consiste en un agent résidant sur un noeud donné du réseau et responsable d'un ou de plusieurs éléments étrangers, éléments avec lesquels il communique via un protocole spécifique.
Dans ce cas, l'agent possède, en plus de sa propre MIB, une MIB décrivant chacun des équipements dont on lui a délégué la responsabilité.
La figure 2.3. illustre les relations qui existent entre un système de gestion, un "proxy-agent" et les éléments étrangers.

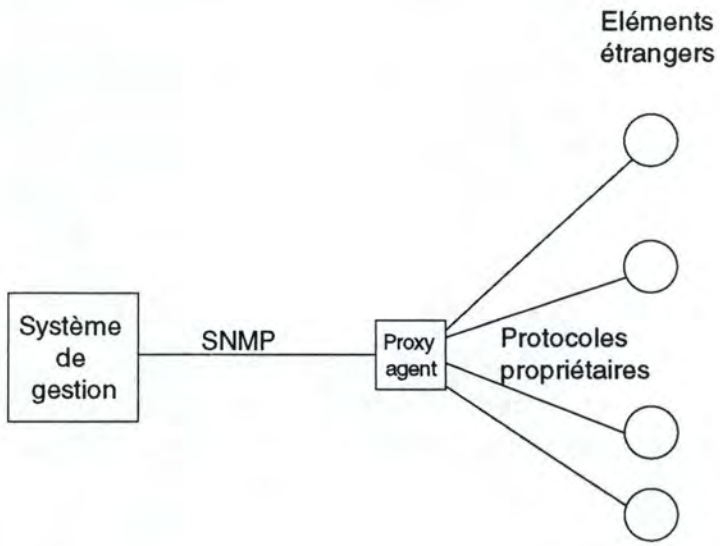


Figure 2.3. Relations entre un système de gestion, un "proxy-agent" et les éléments étrangers.

Voici quelques avantages du mécanisme de délégation:

- permettre au "proxy-agent" d'implémenter une politique d'accès sophistiquée dans laquelle certains sous-ensembles de variables de la MIB sont accessibles par des systèmes de gestion, sans pour autant accroître la complexité de l'élément du réseau;

- permettre la gestion d'éléments du réseau qui ne sont pas adressables via le protocole de transport et le protocole de gestion.
  Quand un élément étranger doit être interrogé, le système de gestion contacte le "proxy-agent" et lui indique l'identité de l'élément étranger. Le "proxy-agent" traduit alors les requêtes envoyées par le système de gestion en des requêtes compréhensibles par l'élément étranger.

La nature de cette traduction varie:

- Si l'élément étranger supporte le protocole de gestion, mais pas le *Service point*-à-point ("connection oriented") utilisé pour transmettre les paquets contenant les données servant à la gestion, le "proxy-agent" a seulement besoin de prendre le contenu des paquets et de l'envoyer à l'élément étranger en utilisant un service de transmission différent.
  Cette approche est peu utilisée car l'expérience a montré qu'il était plus facile d'implémenter les protocoles nécessaires au *Service point*-à-point dans les éléments étrangers et d'éviter l'emploi du "proxy-agent". (Bien sûr, ceci est dû principalement au fait que le protocole SNMP utilisé pour la gestion du réseau est un protocole simple)

- Autrement, si l'élément étranger supporte un protocole de gestion de réseaux différent, le "proxy-agent" agit en tant qu'application-passerelle.

Il existe également une utilisation plus intelligente du "proxy-agent" qui consiste à cacher les informations de gestion. En effet, dans quelques environnements, certaines requêtes concernant la gestion du réseau sont émises fréquemment. Si les réponses ne changent pas aussi fréquemment, un "proxy-agent" pourrait être placé entre l'agent géré et le système de gestion afin de minimiser la charge de l'agent géré, ce dernier ne devant avertir le "proxy-agent" qu'en cas de modification d'une réponse.
Cependant, pour être très efficace, cette approche est difficile à mettre en oeuvre.

La manière dont se passe la liaison entre le système de gestion de réseaux et un "proxy-agent" est laissée au choix de l'implémenteur. Pour bien illustrer comment cette liaison se passe avec le logiciel SPECTRUM, prenons comme exemple la méthode utilisée pour savoir si le conducteur d'un métro ou d'un train ne s'est pas endormi aux commandes de sa locomotive.
En effet, considérons que le "proxy-agent" gère un "conducteur de train" et veille à ce qu'il ne s'assoupisse pas. Dans ce cas-ci, le "proxy-agent" est constitué d'un bouton, d'une lampe et d'une sirène. Le "conducteur de train" est censé pousser toutes les cinq minutes sur le bouton pour indiquer au "proxy-agent" qu'il ne s'est pas endormi. Si, après cinq minutes, le "conducteur du train" n'a pas pressé sur le bouton, le "proxy-agent" va faire clignoter la lampe pour essayer d'attirer son attention. Pour l'instant, le statut du "conducteur de train" pour SPECTRUM est toujours normal.
Si le bouton n'a toujours pas été pressé après dix minutes, le "proxy-agent" va actionner la sirène pour réveiller le "conducteur de train". Le "proxy-agent" va également en informer SPECTRUM qui fera alors passer le statut du "conducteur de train" à un niveau d'alarme mineur.
Enfin, si passé un laps de temps de 15 minutes, le "conducteur de train" n'a toujours pas réagi, on peut alors supposer que quelque chose de grave lui est arrivé (une crise cardiaque par exemple). A ce moment-là, suite à l'information reçue par le "proxy-agent", son statut passera à un niveau d'alarme majeur dans SPECTRUM.

Il revient donc au "proxy-agent" de décider de déclencher des alarmes dans SPECTRUM et de veiller à ce que les agents qu'il gère suivent correctement les règles qui leur ont été imposées.
Dans SPECTRUM, les "proxy-agents" sont appelés des "RBMS-Agents" (Remote Bridge Management System - Agent). Ils utilisent un protocole privé de DEC. Si par exemple, après trois secondes, un bridge ne répond pas à la requête formulée par SPECTRUM, le "RBMS-Agent va déclencher une alarme.

3.4.3. SPECTRUM et le protocole SNMP

Le logiciel SPECTRUM utilise le protocole de gestion de réseaux SNMP pour réaliser toutes les opérations et les requêtes d'informations sur les agents dont il a la responsabilité.
Il permet à l'administrateur du réseau de modéliser, de contrôler et de gérer un équipement particulier ou un groupe d'équipements sur un réseau à partir du SpectroGRAPH.

3.4.4. Un petit mot sur la 2ème version du protocole SNMP

Comme la description de la 2ème version du protocole SNMP (SNMPv2) sort largement du cadre de ce chapitre, nous nous limiterons à montrer les deux principales nouveautés apportées par SNMPv2 [BOBA95] et les avantages que pourrait en tirer le logiciel SPECTRUM. Nous citerons ensuite quelques légères modifications qui ont été apportées à la deuxième version du protocole.

La nouvelle version de SNMP a introduit deux nouvelles primitives importantes: "GET-BULK-REQUEST" et "INFORM-REQUEST".
La première primitive, "GET-BULK-REQUEST", facilite la manipulation d'un volume important de données. Elle minimise le nombre de messages échangés et évite ainsi de surcharger le réseau avec une multitude de messages de gestion. Par exemple, lorsque le système de gestion désire obtenir la valeur de plusieurs variables successives situées dans la MIB d'un agent, il peut le réaliser maintenant en une seule requête au lieu d'envoyer plusieurs requêtes "GET-NEXT-REQUEST" les unes à la suite des autres.
Cet avantage n'est pas négligeable lorsqu'on sait qu'un système de gestion doit permettre une bonne gestion du réseau sans pour cela le congestionner par l'envoi de paquets d'informations relatives à la gestion du réseau (ce qui était pourtant le cas avec le logiciel SPECTRUM lors de notre stage au CERN).

La deuxième primitive, "INFORM-REQUEST", constitue en fait une nouvelle forme de "TRAP" qui est échangée entre systèmes de gestion et à laquelle il est donné réponse. Un nouveau mode d'interaction est donc introduit, c'est l'interaction requête-réponse entre systèmes de gestion. Ce nouveau mode permet en fait une distribution de la gestion parmi les gestionnaires en vue de diminuer la charge du réseau ou de gérer des systèmes de taille plus importante.
Cette nouvelle primitive pourrait être intéressante dans le cas où SPECTRUM serait amené à coopérer avec un autre système de gestion gérant une autre communauté ("community"). Pour prendre un exemple de coopération, considérons le déclenchement d'une alarme qui est, rappelons-le, un dépassement de la valeur seuil d'une variable donnée détecté durant un laps de temps d'observation. Chaque condition d'alarme déclenche un événement (un avertissement sonore et/ou visuel dans le cas de SPECTRUM) et peut à son tour causer une ou plusieurs notifications aux autres systèmes de gestion au moyen d'un "INFORM-REQUEST-PDU".

Outre ces deux nouveautés, nous pouvons également mentionner brièvement quelques modifications mineures apportées par SNMPv2:

- le format général des messages est plus complexe à cause d'un mécanisme de sécurité plus élaboré. SNMPv2 offre une protection contre des modifications de l'information transitant entre le système de gestion et un agent (par exemple, l'information concernant la création d'une communauté). En effet, sans une sécurité efficace, cette information pourrait être interceptée et altérée par un tiers qui aurait alors la possibilité soit d'effectuer des opérations de gestion qui lui sont interdites, soit de falsifier la valeur d'une variable de la MIB d'un agent.

Bien que plusieurs services soient offerts par SNMPv2 pour la sécurité, aucun d'entre eux n'est obligatoire, ils sont juste conseillés. Cette souplesse permet ainsi une compatibilité avec la première version de SNMP;

- le format du "SNMPv2TRAP" a été simplifié. Il utilise maintenant le même format de PDU que les autres opérations;

- le nombre de statuts que peut prendre un PDU a considérablement augmenté, permettant ainsi un bien meilleur diagnostic quant à la cause d'échec.

En ce qui concerne les informations de gestion, SNMPv2 constitue une extension de SNMP. Par conséquent, il est logique que les modules de la MIB définis par SNMP puissent coexister avec le nouveau protocole.
Par contre, les messages échangés dans SNMP et SNMPv2 ne sont pas les mêmes. Afin de permettre une transition vers la nouvelle version aussi douce que possible, il faut prévoir des moyens rendant ces deux protocoles compatibles. Les deux approches retenues sont soit l'utilisation d'un "proxy-agent", soit l'implémentation d'un gestionnaire bilingue.

3.5. Caractéristiques du logiciel SPECTRUM

Dans cette partie, nous allons présenter et décrire les principales caractéristiques du logiciel sur lequel nous avons travaillé durant nos six mois de stage.

3.5.1. SPECTRUM a une architecture modulaire

Le logiciel SPECTRUM est constitué de plusieurs modules. Ainsi, l'administrateur peut n'acheter que les modules qui offrent les fonctionnalités dont il a besoin.
De même, pour éviter d'acheter tous les modules qui lui seront nécessaires, l'administrateur peut aussi développer lui-même ses propres modules dans les langages de programmation C et C++. De cette manière, il dispose d'un logiciel construit "à la carte" et tout-à-fait adapté à son environnement de travail.

3.5.2. SPECTRUM est construit sur base d'une architecture Client/Serveur

Le logiciel SPECTRUM est construit selon une architecture client/serveur orientée-objet. Le SpectroSERVER constitue le serveur et le SpectroGRAPH, le client. Ils communiquent entre eux via le réseau. (voir figure 2.2.)

Grâce à cette construction orientée-objet, les objets appartenant à des classes inférieures héritent leurs attributs et leurs fonctionnalités des classes supérieures, leur fournissant ainsi un comportement de base.

Afin que l'utilisateur puisse travailler avec SPECTRUM par l'intermédiaire du SpectroGRAPH, celui-ci ainsi que le SpectroSERVER doivent avoir été tous les deux préalablement lancés sur des machines non nécessairement distinctes. Dans la plupart des cas, le SpectroSERVER est lancé par l'administrateur du réseau et tourne sur une machine particulière. Quant au SpectroGRAPH, il est lancé depuis la station de travail de l'utilisateur.

Le logiciel SPECTRUM exige une grande quantité de ressources aussi bien pour le client que pour le serveur.
Voici un style de configuration recommandée:

- **Client** : Chaque machine client devrait au moins disposer d'un CPU de type IPC ou IPX avec 48 mégabytes de mémoire RAM, de 120 mégabytes d'espace de "swapping" et d'un disque de 75 mégabytes pour SPECTRUM.

- **Serveur** : Chaque machine serveur devrait disposer de 48 mégabytes de mémoire RAM, de 120 mégabytes d'espace de "swapping" et d'un espace de 75 mégabytes sur disque. En plus, il convient encore de réserver de la place pour la base de connaissances de SPECTRUM et pour les ressources supplémentaires (en mémoire RAM et en espace disque) qui permettent les connexions avec les multiples clients. La machine devra également accéder à un espace pour réaliser périodiquement des sauvegardes de la base de données.

Il est possible de combiner un client et le serveur sur une même machine. Les besoins en ressources deviennent alors de 64 mégabytes de mémoire RAM, de 160 mégabytes d'espace de "swapping" et 150 mégabytes d'espace disque.

Nous pouvons toutefois être étonnés des chiffres que nous avons trouvés dans le document [INT94]. En effet, selon ce document, le client et le serveur auraient les mêmes exigences en mémoire RAM, en espace de "swapping" et en espace-disque. Or, nous avons vu que le serveur, le SpectroSERVER, constituait le système de gestion de réseaux tandis que le client, le SpectroGRAPH, jouait simplement le rôle d'interface graphique pour communiquer avec le SpectroSERVER. Et à notre avis, une interface graphique exige beaucoup moins de ressources qu'un système de gestion de réseaux.

3.5.3. SPECTRUM offre une bonne perception des problèmes pouvant survenir sur le réseau

Tous les équipements du réseau sont représentés par des icônes dont la couleur de fond dépend de leur statut:

- **vert** : aucun problème, le contact est établi avec l'équipement;
- **bleu** : condition initiale. Aucun contact avec l'équipement depuis que le modèle a été créé;
- **gris** : l'équipement ne peut être atteint à cause d'une condition d'erreur inconnue dans un équipement intermédiaire situé entre le SpectroSERVER et l'équipement représenté par l'icône de couleur grise;
- **jaune** : alarme mineure. Premier niveau d'opération marginale. Cette couleur est aussi utilisée pour indiquer qu'un modèle pourrait avoir une adresse IP dupliquée;
- **orange** : alarme majeure. Second niveau d'opération marginale. Cette couleur peut également indiquer un problème de gestion au niveau de l'équipement;
- **rouge** : perte de contact avec un équipement.

Lorsqu'un équipement change d'état, la couleur de fond de l'icône le représentant change également et une alarme sonore peut se faire entendre. Ainsi, l'attention de l'opérateur du réseau se porte tout de suite sur le problème critique.

Cependant, le logiciel SPECTRUM n'est pas prévu pour détecter les petites pannes journalières du réseau à cause de la richesse des vues, des menus et du multi-fenêtrage. Pour exécuter des tâches telles que l'identification d'un problème, il faut se trouver soit dans une vue "*Topologie*" du réseau qui modifie la couleur de fond de l'icône représentant l'équipement défectueux, soit dans la vue "*Alarme*" grâce à laquelle l'opérateur peut obtenir de plus amples informations sur l'alarme.
Ces deux types de vue seront décrites dans la partie 3.5.4. qui suit.

3.5.4. SPECTRUM offre différentes visions du réseau et de ses composants

Le SpectroGRAPH fournit un ensemble de vues au travers desquelles l'utilisateur peut examiner le réseau. Il existe plusieurs types de vues qui fournissent chacune une perspective unique de gestion du réseau. Ces vues sont utilisées pour représenter les différentes hiérarchies du réseau, les activités du réseau et les informations sur un équipement particulier du réseau.

L'utilisateur peut employer plusieurs techniques pour accéder aux différentes vues offertes par le SpectroGRAPH. La plus directe consiste à y accéder par l'intermédiaire d'une commande trouvée dans un menu de SPECTRUM. Une fois la vue ouverte, il peut également maximiser ou minimiser la fenêtre.
Par exemple, il a la possibilité d'observer le réseau à partir d'une perspective représentant des lieux pour localiser un de ses composants (voir figure 2.4. & 2.7.). Ensuite, il peut passer à la perspective topologique (voir figure 2.5.) et distinguer tous les autres composants qui y sont connectés. Enfin, il peut regarder le réseau d'un point de vue organisationnel (voir figure 2.6.) pour prendre connaissance des groupes de travail qui s'occupent d'un composant.

Chaque vue de SPECTRUM a deux modes d'opérations:

- Le mode "**navigation**" : il s'agit du mode utilisé pour se déplacer normalement d'une vue à l'autre. Dans ce mode, une icône représente un équipement du réseau, un sous-réseau, un segment, une localisation physique...;

- Le mode "**édition**" : ce mode est utilisé pour effectuer des changements dans une vue tels qu'ajouter, effacer ou modifier un équipement du réseau. Ici, une icône est simplement un objet graphique que l'utilisateur peut manipuler.

Les différents types de vues de SPECTRUM :

Le logiciel SPECTRUM organise et présente ses informations à l'aide de quatre grands types de vues, à savoir les vues "*Localisation*", les vues "*Topologie*", les vues "*Organisation*" et les vues donnant des informations sur les composants du réseau. Ces types peuvent être divisés en deux classes suivant la manière dont ils présentent les informations: la classe hiérarchique et la classe non-hiérarchique.

a) Les différentes vues hiérarchiques

Les trois types de vue que nous allons vous présenter sont hiérarchiques dans le sens où l'on part du niveau de détail le plus haut, le plus général, pour se diriger vers le niveau de détail le plus bas, le plus spécifique.

- les vues "*Localisation*" constituent des représentations physiques du réseau qui correspondent à des lieux: une partie du monde, un pays, une région, un bâtiment, une salle, une pièce, etc...
  La figure 2.4. illustre un exemple de vue "*localisation*". Il s'agit d'une vue sur une partie du monde.

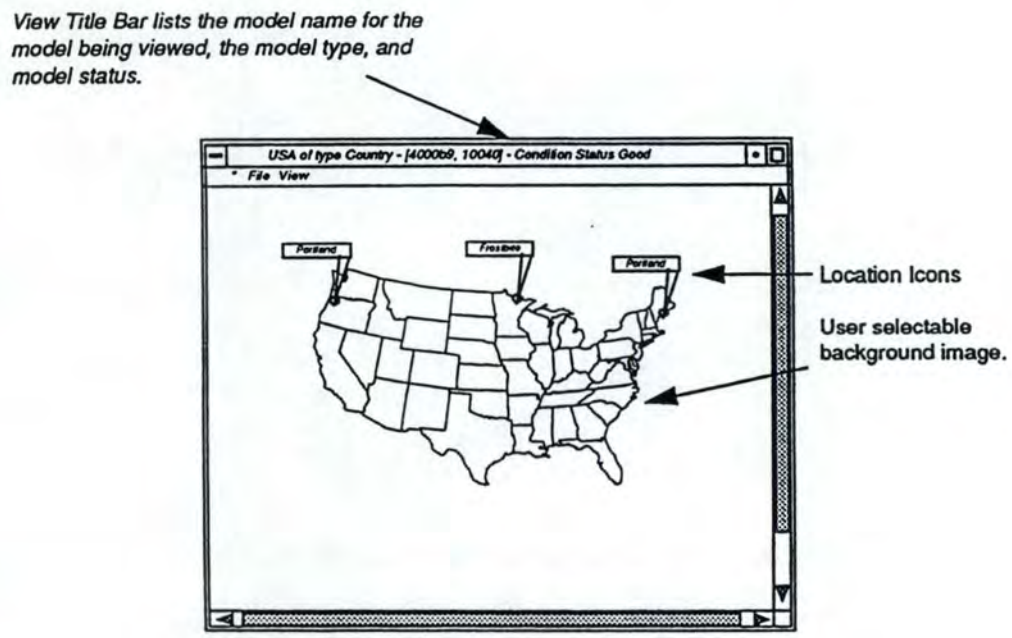


Figure 2.4. Vue "*localisation*" au niveau mondial

- les vues "*Topologie*" du réseau sont des représentations du réseau basées sur les connexions des équipements du réseau. Elles contiennent des icônes représentant des appareils (bridges, routeurs, stations de travail...), des sous-réseaux, des segments et les connexions existant entre eux.

  Par exemple, lorsqu'un opérateur se trouve au niveau le plus haut dans la hiérarchie des vues "*Topologie*", il dispose de la représentation du "back-bone" du CERN. S'il désire la description détaillée de l'un des seize sous-réseaux en fibre optique composant le "back-bone" du réseau, il peut aller cliquer sur l'icône représentant le sous-réseau désiré. Il obtient alors une nouvelle vue qui comprend tous les appareils (bridges, routeurs...) connectés à ce sous-réseau ainsi que les segments qui leur sont reliés.

  La hiérarchisation des vues "*Topologie*" permet de descendre jusqu'au niveau des machines et des stations de travail appartenant à un segment. Cependant, ce niveau de détail n'est pas d'application au CERN qui n'en ressent pas l'utilité.
  La figure 2.5. illustre un exemple de vue "*Topologie*".

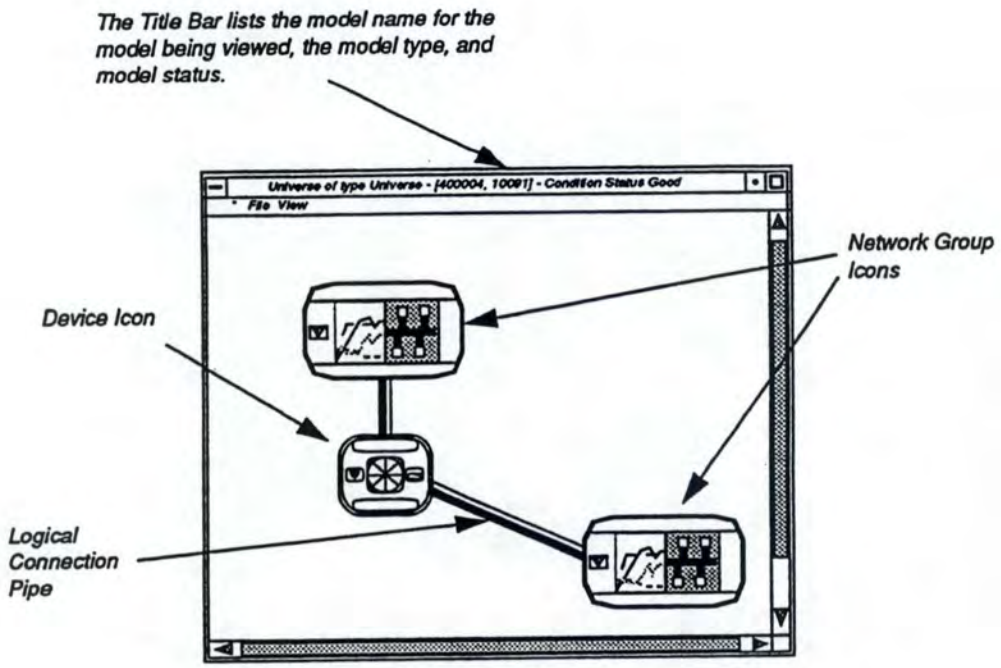


Figure 2.5. Une vue "*Topologie*".

- les vues "*Organisation*" donnent des représentations reposant sur la structure d'organisation. Elles sont surtout utiles quand on éprouve le besoin de savoir quels effets pourraient avoir une panne ou une reconfiguration d'un réseau sur les différents départements de l'entreprise, les groupes de travail ou les individus.
  La figure 2.6. illustre un exemple de vue "*Organisation*".

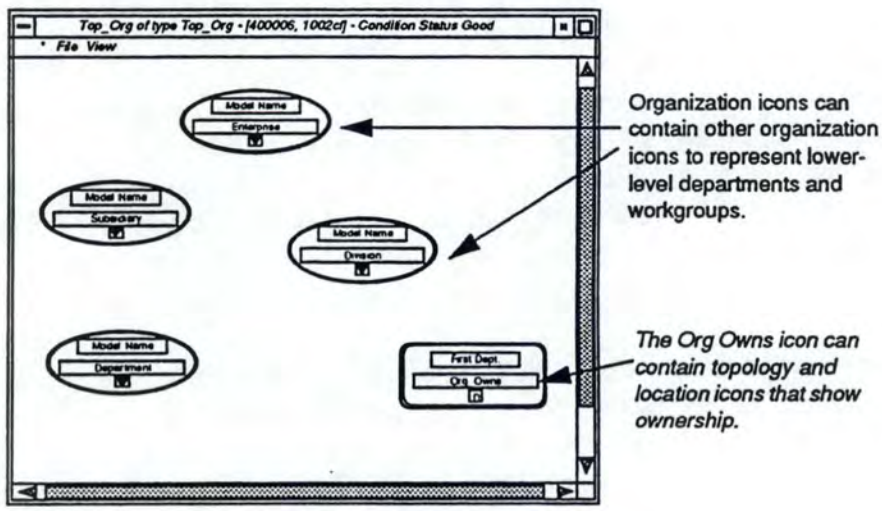


Figure 2.6. Une vue "*Organisation*"

La figure 2.7. illustre un exemple de hiérarchisation. Il s'agit d'une hiérarchie de vues "*Localisation*" qui part d'un niveau mondial constituant la racine de la hiérarchie et qui passe par

toute une série de niveaux intermédiaires pour arriver au niveau le plus bas de la hiérarchie constitué par des compartiments de salles.

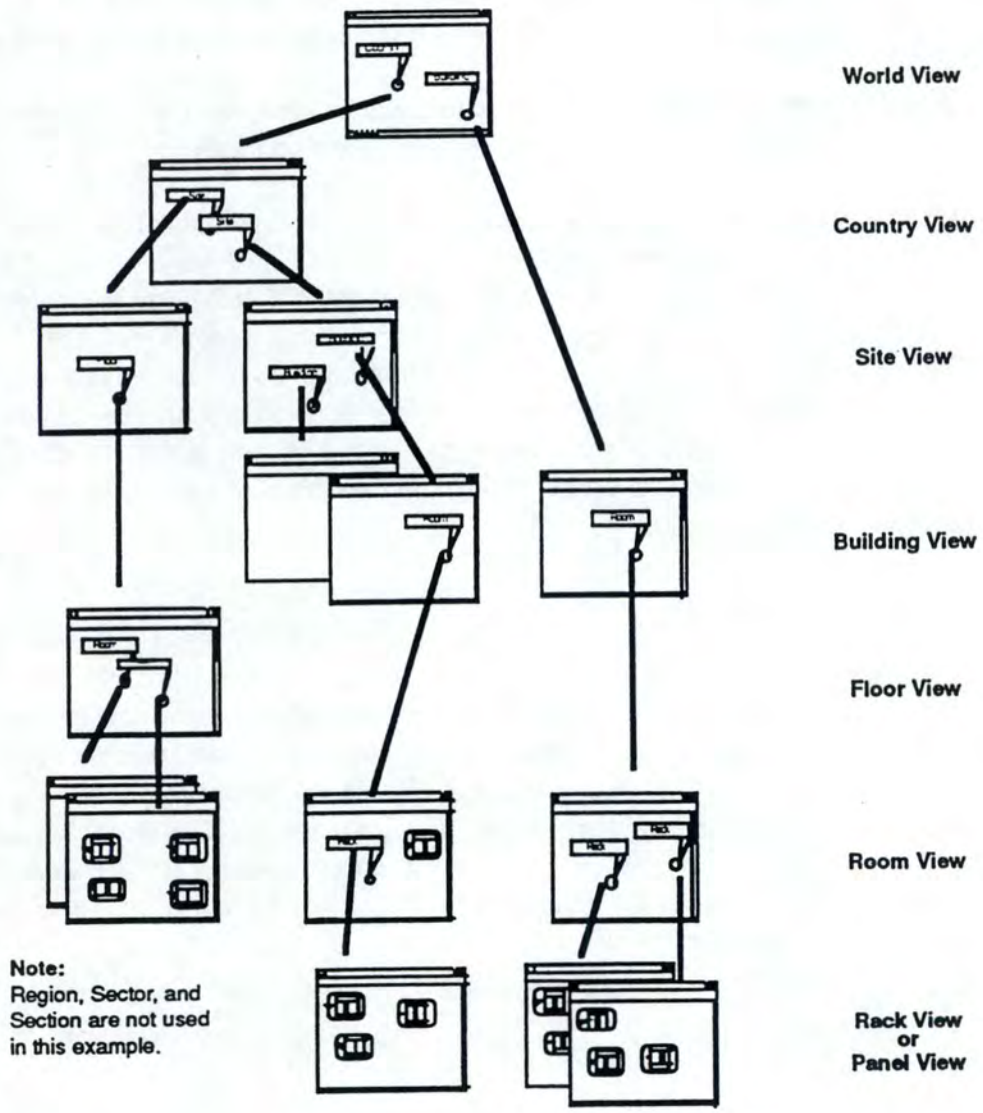


Figure 2.7. Une vue "*Localisation*" hiérarchique.

De plus, certaines vues (les vues "*Topologie*", les vues "*Localisation*",...) permettent d'accéder à plusieurs sous-vues qui contiennent de plus amples informations sur les équipements du réseau. La plupart de ces sous-vues sont associées à une icône-bouton sur laquelle il suffit d'aller cliquer pour les ouvrir. Cette icône-bouton est située à l'intérieur de l'icône représentant un équipement.

Voici quelques exemples de sous-vues:

- la vue "*Configuration*" permet l'initialisation des valeurs pour un équipement du réseau: son nom, son type, son adresse physique, sa localisation physique...;

- la vue "*Cablewalk*" illustre les connexions qui existent entre un équipement du réseau, tel qu'un bridge, et les sous-réseaux auxquels il est connecté;

- la vue "*Information*" donne des informations sur un équipement du réseau: son nom, le sous-réseau auquel il appartient, son statut, le fait qu'il fasse l'objet d'un polling, les valeurs limites des attributs à partir desquelles SPECTRUM doit déclencher une alarme (nombre d'erreurs par seconde, nombre de paquets par seconde,...), etc...
  Par exemple, à partir d'une vue "*Topologie*" qui comprend divers équipements du réseau, l'opérateur peut obtenir des informations détaillées sur l'un de ces équipements particuliers. Il lui suffit d'aller cliquer sur l'icône-bouton adéquate située dans l'icône représentant l'équipement en question pour que s'ouvre alors une vue "*Information*" contenant les informations.

Nous avons vu au point 3.5.3. que les icônes utilisées pour représenter les divers équipements apparaissent dans des couleurs différentes en fonction de l'état ou du statut des équipements: vert, bleu, gris, jaune, orange et rouge. Lorsqu'un équipement change d'état (par exemple, il ne répond plus au système de gestion ou bien une valeur seuil d'un de ses attributs a été dépassée) la couleur affichée à l'intérieur de l'icône change également et se met à clignoter pour bien attirer l'attention de l'utilisateur. Pour arrêter ce clignotement, l'utilisateur doit se servir de la commande "Acknowledge". Au cas où l'utilisateur éprouve des difficultés à discerner les différentes couleurs, il peut lancer un script qui lui donnera des indications auditives lors d'un changement d'état d'un équipement du réseau.

b) Quelques vues non-hiérarchiques

- la vue "*Alarme*" affiche la liste de toutes les alarmes actuelles du réseau en donnant une brève description de chaque alarme. En allant se positionner sur l'une d'entre elles, l'utilisateur peut également observer l'icône représentant l'équipement dont provient l'alarme. En fonction de sa gravité, l'alarme sera affichée dans une couleur différente.
  Cette vue permet également, en cliquant sur l'alarme en question, d'accéder à la liste des causes probables de l'alarme, aux solutions envisageables ainsi qu'aux personnes à contacter pour y remédier.
  La figure 2.8. illustre la fenêtre du SpectroGRAPH contenant toutes les alarmes en cours et leur état.

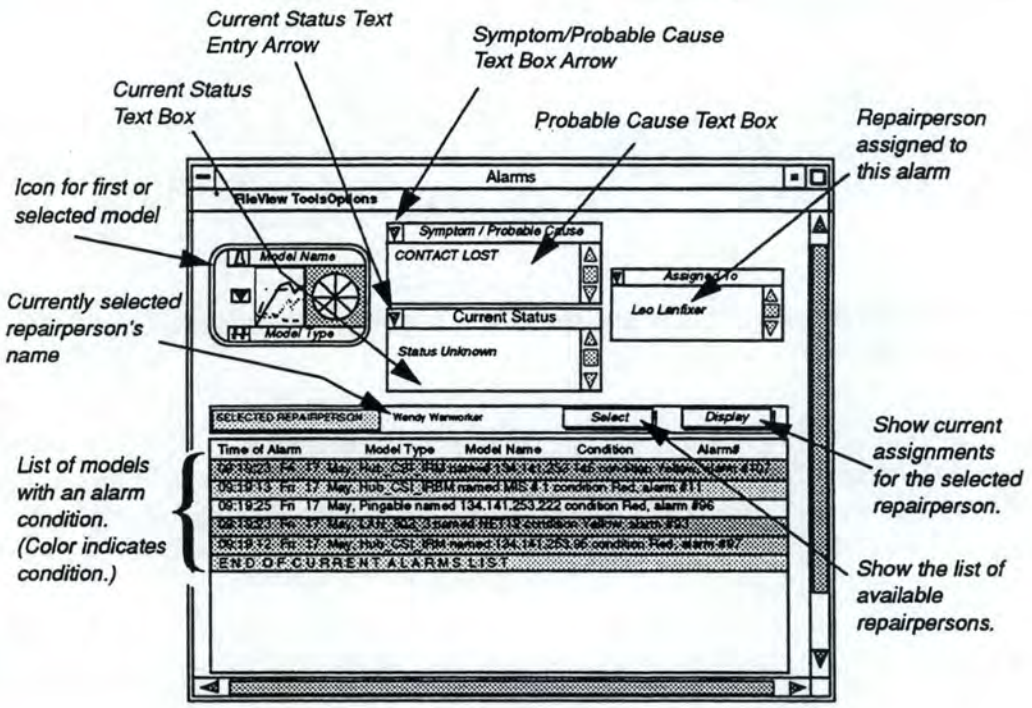


Figure 2.8. La vue "*Alarme*" du SpectroSERVER.

Puisque le traitement des pannes pouvant survenir sur le réseau constitue une activité essentielle dans la gestion des réseaux, l'utilisateur a plutôt intérêt à laisser cette vue "*Alarme*" ouverte la plupart du temps. Cependant, lorsqu'une alarme change d'état alors que la vue "*Alarme*" est icônisée sous la forme d'un réveil, l'icône se met à clignoter et une sonnerie retentit. Ce phénomène est illustré à la figure 2.9.;

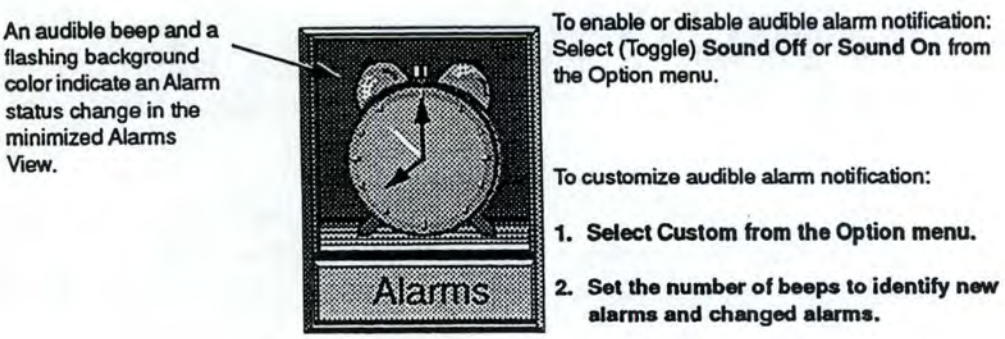


Figure 2.9. La vue "*Alarme*" icônisée.

- la vue "*Evénement*" présente une sorte de bloc-notes reprenant tous les événements du réseau détectés par le logiciel SPECTRUM depuis sa mise en service: le lancement du SpectroSERVER et des sessions du SpectroGRAPH, l'établissement ou la perte d'un contact avec un équipement du réseau, le changement de statut d'un équipement du réseau...;

- la vue "*Recherche*" constitue un outil permettant à l'utilisateur de rechercher un équipement du réseau d'après une de ses caractéristiques: son nom, son type, son adresse physique...

3.5.5. SPECTRUM permet la gestion d'une grande variété de produits différents utilisés dans les réseaux

La recherche de tous les équipements du réseau fabriqués par la société Cabletron est très facile. Lors de l'initialisation du SpectroSERVER, ils sont retrouvés automatiquement par la fonction "Auto-Discovery" offerte par SPECTRUM. Tous les autres équipements provenant de divers vendeurs sont considérés par cette fonction comme des équipements de type "générique SNMP", pour autant qu'ils supportent le protocole SNMP.
Cette fonction permet un gain de temps considérable dans le sens où elle évite aux opérateurs de devoir introduire manuellement tous les équipements du réseau dans la base de données de SPECTRUM.
Cependant, le logiciel SPECTRUM n'est pas très facile à installer dans un nouvel environnement. En effet, il offre un grand nombre de facilités mais qui sont seulement réservées aux équipements fabriqués par l'usine-mère, Cabletron.
En ce qui concerne les équipements fabriqués par d'autres vendeurs, l'utilisateur peut acheter ou créer ses propres modèles. La création de modèles génériques SNMP s'effectue sans encodage. Il suffit de choisir les points de dérivation dans la hiérarchie des classes en éditant quelques fichiers et en exécutant quelques scripts.

3.5.6. SPECTRUM fait appel à une technologie de modélisation inductive

Le logiciel SPECTRUM est doté d'un mécanisme d'intelligence artificielle qui fait appel à une technologie de modélisation inductive. Cette modélisation, couplée avec la conception orientée-objet du logiciel, lui permet de comprendre les dépendances qui pourraient exister entre deux entités du réseau.

L'exemple de ce mécanisme d'intelligence que nous avons trouvé dans [SPEC5] ne nous semble pas constituer la preuve d'une très grande intelligence de la part de SPECTRUM. Néanmoins, comme cet exemple est le seul que nous aillons pu trouver, nous l'avons quand-même illustré à la figure 2.10.
Au départ, prenons l'hypothèse que tous les appareils aient répondu "présents" aux différents "pollings" effectués par SPECTRUM (la couleur initiale de toutes les icônes est donc verte).

A un moment donné, l'appareil qui est entouré sur la figure 2.10. ne répond plus. Dès lors, l'icône "parente" symbolisant le sous-réseau comprenant cet appareil change d'état et devient de couleur orange. De la même sorte, puisque SPECTRUM n'arrive plus à joindre les composants directement reliés à l'appareil occasionnant un problème, la couleur des icônes les symbolisant devient grise (équipement impossible à atteindre).

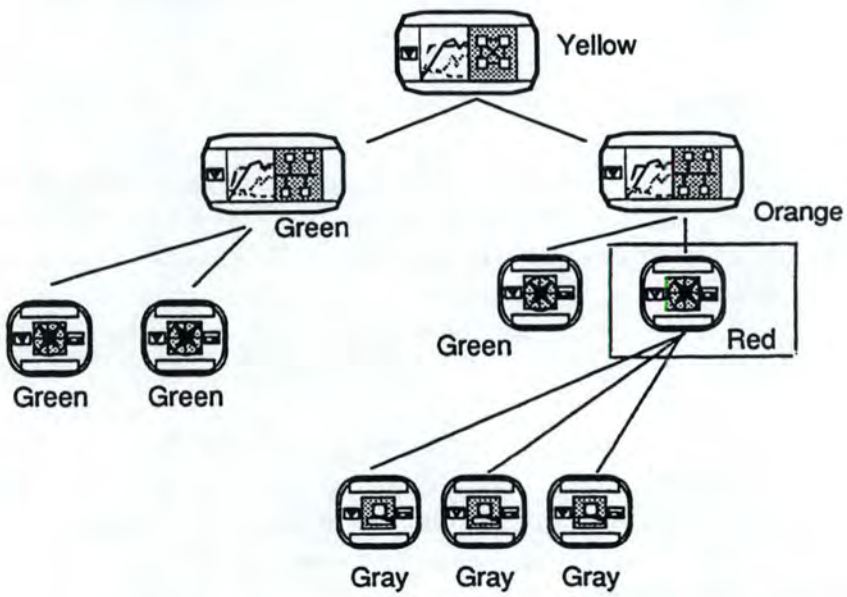


Figure 2.10. Changement de la couleur des icônes suite à la survenance d'un problème au niveau d'un appareil.

Le logiciel SPECTRUM peut ainsi examiner plusieurs événements qui se produisent sur le réseau et tirer la conclusion qu'ils proviennent tous du même problème tel qu'un bridge défectueux.

Cette intelligence que l'on retrouve dans le logiciel lui permet également de réduire extrêmement le nombre de messages à afficher dans la vue "*Alarme*" indiquant des problèmes sur le réseau. Dans l'exemple de notre bridge défectueux, seul le message concernant son problème est affiché. Tous les autres problèmes qui constituent une conséquence logique de la panne du bridge ne sont pas pris en compte.

3.5.7. SPECTRUM offre une interface graphique conviviale

Pour tous les terminaux ou stations de travail qui acceptent une interface graphique, le logiciel SPECTRUM utilise le SpectroGRAPH pour dialoguer avec les administrateurs et les opérateurs qui s'occupent de la gestion du réseau.

Il s'agit d'une interface basée essentiellement sur l'utilisation de la souris avec des menus déroulants, des "popup menus", des sous-menus, des icônes, des boîtes de dialogue, etc...
A l'aide de cette interface graphique, chaque utilisateur a la possibilité d'adapter son environnement de travail en créant ses propres icônes et des raccourcis-clavier;

L'année passée, nous avons suivi le cours d'interface homme-machine donné par Monsieur François Bodart, Professeur à l'Institut d'Informatique des Facultés Notre-Dame de la Paix à Namur. Nous allons maintenant nous en servir pour critiquer l'interface de SPECTRUM:

Quelques points positifs :

- L'utilisation de la souris permet un déplacement rapide à travers l'écran. Pour les personnes éprouvant des difficultés à manipuler la souris, le "double clique" est paramètrable, c'est-à-dire qu'on a la possibilité de déterminer le temps séparant les deux pressions successives sur un bouton de la souris. A côté de cela, nous trouvons également des raccourcis-clavier qui permettent d'accroître la vitesse d'exécution des opérations;

- Les menus sont bien structurés: les items les plus importants se situent en haut ou à gauche et au fur et à mesure que l'importance d'un item diminue, il se situera vers le bas ou vers la droite. De plus, pour accroître la rapidité d'exécution des tâches, il est possible de laisser les menus affichés à l'écran pour éviter de devoir l'ouvrir chaque fois qu'on en éprouve le besoin;

- L'interface graphique dispose d'un bon assortiment de couleurs douces et agréables à regarder telles que le jaune, le bleu, le vert doux, le rouge non strident... Elle n'utilise pas de couleurs vives et irritantes pour l'oeil humain. Par exemple, tous les menus sont écrits en jaune sur un fond bleu, ce qui en facilite la lecture;

- Pour attirer l'attention de l'utilisateur sur un problème particulier, l'interface fait encore appel aux couleurs ainsi qu'à leur clignotement (voir le point 3.5.3.). Pour les personnes ayant du mal à discerner les différentes couleurs, SPECTRUM emploie une "sonnerie" dont la mélodie varie en fonction du type de problème;

- Les icônes expriment assez bien ce qu'elles représentent, ce qui réduit au maximum la distance sémantique. Par exemple, dans la vue "*Topologie*", on peut voir la représentation réelle (d'une partie) du réseau avec ses bridges, ses routeurs, ses machines, ses câbles, les connexions existantes...

Quelques points négatifs :

- Lorsqu'un utilisateur souhaite maximiser une fenêtre du SpectroGRAPH pour observer toutes les informations qu'elle contient, sans être obligé de jouer avec les ascenseurs

situés à droite et en-dessous de la fenêtre, celle-ci prend une taille maximale fixe qui n'est pas proportionnelle à la taille de l'écran de l'utilisateur. Cet inconvénient est surtout ennuyeux pour les vues comprenant beaucoup d'icônes telles que les différentes vues "*Topologie*" et "*Localisation*";

- Si deux mêmes vues sont ouvertes sur deux SpectroGRAPHs différents au même moment (par exemple deux vues "*Topologie*") et qu'un utilisateur modifie la première grâce au mode d'édition, l'autre ne sera pas modifiée automatiquement. Pour que les changements opérés dans la première vue soient pris en compte dans la deuxième vue, il faut fermer et ré-ouvrir cette dernière;

- La zone de "cliquage" sur certaines icônes-boutons est assez réduite et n'est pas bien définie. Il revient à l'utilisateur d'apprendre par des essais répétés l'endroit précis où il doit cliquer pour obtenir le résultat voulu.

Nous venons de décrire quelques atouts et quelques inconvénients de l'interface graphique du logiciel SPECTRUM. Nous pourrions continuer à la critiquer plus longuement, mais nous croyons que cette critique plus poussée dépasserait le cadre de ce mémoire. C'est pour cette raison que nous nous en tenons à ce qui vient d'être écrit.

3.5.8. SPECTRUM permet de visionner un équipement particulier et ses performances

Pour observer les performances d'un équipement tel qu'un bridge ou un routeur, il suffit à l'utilisateur de cliquer sur une icône-bouton située à l'intérieur de l'icône représentant l'équipement. Cette action aura pour effet d'ouvrir une vue "Performance" qui donnera diverses statistiques sur les performances de l'équipement sous la forme de graphiques ainsi que des valeurs clés pour son contrôle.

La figure 2.11. illustre la fenêtre représentant la vue "Performance" dont l'utilisateur dispose pour observer les performances d'un équipement particulier.

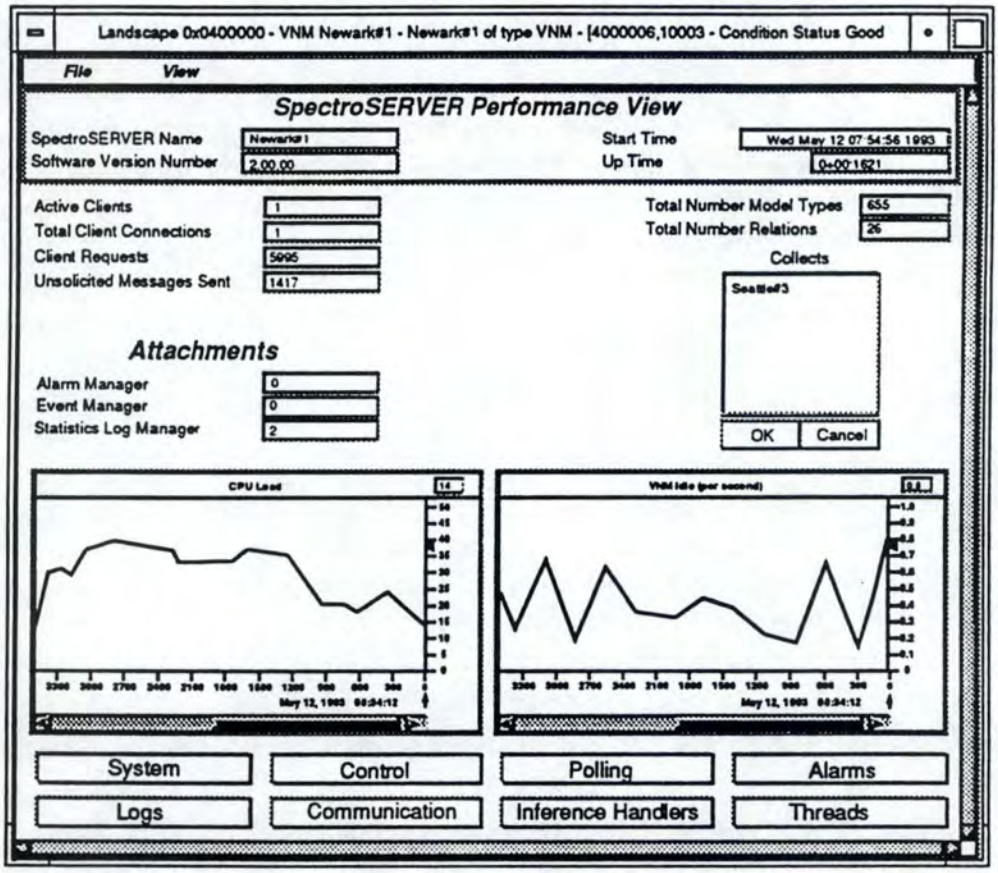


Figure 2.11. La vue "performance" sur un équipement particulier.

3.5.9. SPECTRUM offre une "Command Line Interface" (CLI) permettant d'interagir avec le SpectroSERVER sans utiliser le SpectroGRAPH

Le "*Command Line Interface*" fournit un moyen d'accéder au SpectroSERVER dans des situations où il n'est pas possible ou pas désirable d'utiliser le SpectroGRAPH. On se trouve face à une telle situation lorsque le terminal utilisé est orienté caractère.
Le CLI fournit un ensemble de commandes permettant à l'utilisateur d'exécuter des opérations qui, autrement, ne peuvent être exécutées seulement que par l'intermédiaire du SpectroGRAPH.
Les commandes du CLI peuvent être introduites dans des "shells scripts" pour donner à l'utilisateur une interface encore plus puissante et plus versatile.

Cette caractéristique sera développée plus en profondeur dans le quatrième chapitre de ce mémoire consacré à la présentation de notre premier projet.

3.5.10. SPECTRUM est en mesure d'assigner des techniciens à un problème

La vue "*Réparation*" permet d'assigner une personne à un type de pannes pour les résoudre rapidement. Quand l'utilisateur ouvre la vue "Réparation", il aperçoit sous la forme d'icônes tous les dépanneurs disponibles. En allant cliquer sur l'une de ces icônes, l'utilisateur peut voir quelles sont les pannes qui sont assignées à un dépanneur.

Ainsi, lorsqu'une panne est détectée sur un équipement du réseau, SPECTRUM peut envoyer un avertissement par courrier électronique à un des dépanneurs en fonction du type de la panne. Celui-ci peut alors intervenir rapidement.

3.5.11. SPECTRUM offre une bonne gestion de la base de données située dans sa base de connaissances

Ce que nous entendons ici par la base de connaissances du logiciel, c'est l'ensemble formé par les trois éléments principaux de son architecture logique: le SpectroSERVER, les différents SpectroGRAPH et applications qui servent à interagir avec le SpectroSERVER et la base de données avec ses utilitaires.

La base de données de SPECTRUM est située dans le SpectroSERVER qui s'occupe de la gérer. Elle contient un catalogue comprenant tous les modèles et toutes les relations entre ces modèles existant dans SPECTRUM et qui correspondent à la structure de toute l'information du réseau. La base de données permet également de stocker toutes les configurations spécifiques des équipements du réseau, toutes les statistiques et tous les événements. A cette fin, la base de données est divisée en deux parties: une base de données active qui contient les informations les plus récemment acquises et des fichiers d'archivages dans lesquels sont transférées les informations plus anciennes.

Quand un utilisateur ouvre une vue avec des icônes représentant des équipements du réseau, l'information présentée provient d'un modèle ou d'une simulation d'un équipement réel qui se trouve dans la base de données. En y modélisant les divers équipements, SPECTRUM peut fournir un grand nombre d'informations à propos des équipements du réseau ainsi que de sa structure, même lorsqu'on a perdu le contact avec un équipement.

L'information se trouvant dans la base de données sur les équipements spécifiques provient de plusieurs sources. Les types de modèles, les règles, les relations et les attributs sont définis grâce à l'éditeur des types de modèle (MTE). Les modèles et les associations sont définis à partir du SpectroGRAPH ou toute autre application client. L'information qui doit être gardée entre deux exécutions du SpectroSERVER est également stockée dans la base de données.
Le SpectroGRAPH ou l'application client accède indirectement à la base de données en interrogeant le SpectroSERVER grâce à des appels au "SpectroSERVER Application Programmer Interface" (SSAPI). Grâce à cette interface, le développeur d'une application client ne doit pas connaître les détails d'implémentation de la base de données.

La base de données de SPECTRUM est accessible à trois niveaux:

- le système de gestion de la base de données (DBMS) constitue le niveau le plus bas. Il est accessible par le SpectroSERVER et possède un ensemble limité d'outils pour accéder à la base de données;

- le SpectroSERVER permet d'accéder à la couche de modélisation de la base de données pour le SpectroGRAPH, aux applications, et aux autres caractéristiques du SpectroSERVER telles que le gestionnaire des événements, le gestionnaire des statistiques...;

- au niveau le plus haut, on trouve les applications spécifiques qui fournissent une interface-utilisateur pour permettre à l'utilisateur d'accéder aux informations de la base de données.
  Le SpectroGRAPH et l'utilitaire "AlarmMonitor" qui s'occupe de gérer les alarmes survenant sur le réseau font partie de ces applications spécifiques.

Pour éviter que plusieurs utilisateurs ou applications ne modifient en même temps des éléments de la base de données, les accès concurrents ne sont pas autorisés. Pour ce faire, lorsqu'un utilisateur accède en écriture à la base de données, le logiciel SPECTRUM pose un "verrou" empêchant tout autre accès simultané.

Le logiciel SPECTRUM propose également un niveau de sécurité pour les accès à la base de données. Ainsi, n'importe quel utilisateur ne pourra pas accéder à toute l'information mais seulement à une partie, ce qui veut dire que les administrateurs du réseau ont le contrôle sur ce que peut observer à l'écran un utilisateur individuel. Cela leur permet par conséquent de limiter les accès de chaque utilisateur en fonction de leurs responsabilités dans l'organisation.
Cependant, puisqu'il n'existe qu'un seul niveau de privilège administratif, l'administrateur d'une quelconque partie du réseau peut accéder au profil des utilisateurs s'occupant des autres parties du réseau. La figure 2.12. donne une représentation de l'architecture logique du logiciel SPECTRUM.

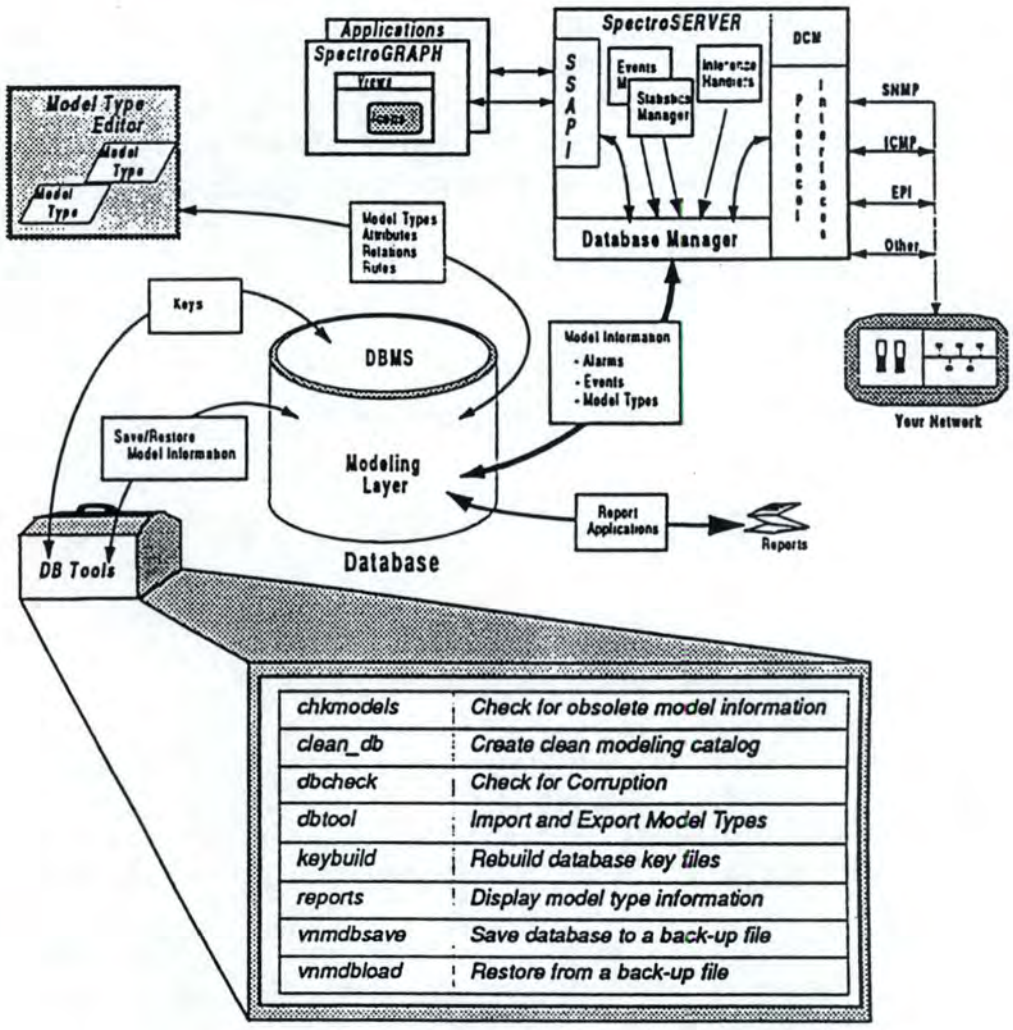


Figure 2.12. Architecture logique du logiciel SPECTRUM.

Nous allons maintenant détailler brièvement les différents composants de chacun des trois éléments principaux de l'architecture logique de SPECTRUM.

a) Le SpectroSERVER et ses composants

Les composants que comprend le SpectroSERVER:

- le *SSAPI* est une interface par laquelle le SpectroGRAPH ou une application client interroge indirectement la base de données;

- le "*Event Manager*" constitue le module qui gère tous les événements se produisant au niveau du réseau: le démarrage du SpectroSERVER et des sessions des SpectroGRAPH, l'établissement ou la perte de contact avec un équipement, les changements de statut d'un équipement...;

- le "*Statistic Manager*" s'occupe de la gestion de toutes les valeurs des variables de la MIB des agents qui ont été obtenues par le mécanisme de "polling";

- le "*Inference Handlers*" représente le module "intelligent" du SpectroSERVER. Il contient des règles d'inférence à partir desquelles SPECTRUM déduit les dépendances entre équipements du réseau;

- le "*Protocol Interface*" permet au SpectroSERVER de dialoguer avec les différents composants du réseau;

- le "*Database Manager*" se charge de gérer la base de données et de répondre aux requêtes émises par le SpectroSERVER.

b) Les outils permettant d'interagir avec le SpectroSERVER

Voici les deux outils nécessaires pour dialoguer avec le SpectroSERVER:

- le *SpectroGRAPH* constitue, comme nous l'avons déjà dit, une interface graphique permettant de dialoguer avec le SpectroSERVER par l'intermédiaire de différentes vues;

- les *applications* représentent des programmes ou des utilitaires qui peuvent directement interroger le SpectroSERVER grâce au SSAPI, sans passer par un SpectroGRAPH.

c) La base de données et ses utilitaires

Voici une explication des différents éléments ayant un rapport direct avec la base de données de SPECTRUM:

- la "*database*" même reprend toutes les informations du logiciel SPECTRUM sur les modèles symbolisant des composants du réseau avec leurs attributs et leurs relations, sur les alarmes et sur les événements. Elle constitue la base de données active de SPECTRUM de sorte que toutes les informations qu'elle contient sont récentes et n'y restent que durant une période de temps limitée;

- les "*Report Applications*" créent des rapports sur les informations contenues dans la base de données à destination des opérateurs et administrateurs du réseau;

- le "*Model Type Editor*" (MTE) offre à l'utilisateur la possibilité de créer de nouveaux types de modèle qui sont encore inconnus dans SPECTRUM;

- les "*DB Tools*" met à la disposition de tout utilisateur des outils permettant de sauver les informations de la base de données sur des fichiers d'archivages ou de les récupérer. Il offre également des outils pour vérifier l'intégrité de la base de données et le risque de corruption, pour faire des "back-up" de la base de données, pour permettre aux "Report Applications" de générer des rapports...

3.5.12. SPECTRUM utilise le mécanisme de "polling"

Le SpectroSERVER permet deux types de "polling" pour un appareil:

- le "polling" automatique qui est à l'initiative du SpectroSERVER;

- le "polling" manuel qui est à l'initiative de l'opérateur.

Pendant le "polling" automatique, le SpectroSERVER vérifie le statut des appareils et regarde les valeurs pour les variables spécifiées dans la MIB des agents.
A cette fin, il fabrique, suivant le protocole de gestion de réseaux SNMP, des paquets contenant les demandes de requêtes et la suite de numéros indiquant l'endroit dans l'arborescence où se situe la variable à atteindre pour un équipement. Ces paquets sont ensuite envoyés via le réseau vers l'équipement désiré toujours suivant le protocole de gestion.

C'est le rôle de l'administrateur du réseau de spécifier au SpectroSERVER quels sont les appareils qui doivent faire l'objet d'un "polling", les intervalles de temps entre ces "pollings" et les variables de la MIB à vérifier.

L'éditeur du bloc d'informations génériques (GIB) de SPECTRUM permet à un utilisateur de générer une fenêtre grâce à laquelle il peut entamer un "polling" manuel sur un appareil spécifié. Un "polling" lancé par un opérateur entraîne une mise à jour de l'état de l'information de la base de données du SpectroSERVER exactement de la même manière que l'effectuerait un "polling" automatique. Ceci implique que si une valeur d'une variable de la MIB provenant d'un "polling" manuel dépasse une valeur limite, une alarme est instantanément déclenchée, exactement comme s'il s'agissait d'un "polling" automatique.

Le logiciel SPECTRUM permet également l'utilisation du mécanisme de trappes. Celles-ci sont enregistrées par SPECTRUM comme des événements. Cependant, le CERN ne se sert pas encore de ce mécanisme.

Les informations ainsi recueillies sont stockées dans la base de données active de SPECTRUM gérée par le SpectroSERVER. Elles peuvent être vérifiées à partir des vues qu'offre le logiciel.

Passé un certain laps de temps, ces données sont alors automatiquement transférées dans des fichiers d'archivages.

4. Les atouts du logiciel SPECTRUM qui lui ont permis d'être choisi parmi les différents logiciels de gestion

Pour poursuivre, nous allons donner quelques atouts de SPECTRUM qui lui ont permis d'être choisi par le responsable du groupe CS ("Communications Systems") de la division CN du CERN, parmi tous les logiciels de gestion de réseaux présents sur le marché.

Comme nous l'avons déjà mentionné, le CERN utilise depuis plusieurs années un système de gestion de réseaux appelé CAW qu'il a développé lui-même pour répondre à ses besoins spécifiques, puisqu'à l'époque, il n'existait pas encore de produits équivalents sur le marché. Par conséquent, le CERN avait une idée très claire de ce qu'il attendait de la part d'un logiciel de gestion de réseaux commercial, tout en sachant bien que certaines fonctions qu'offrait CAW étaient assez "spécifiques au site". Toutes ces connaissances en matière de gestion de réseaux que possédait déjà le CERN influencèrent fortement le choix du responsable du groupe CS parmi les logiciels commerciaux qui lui ont été présentés.

Voici les caractéristiques principales que recherchait le CERN dans un système de gestion:

- de bonnes possibilités d'intégration pour pouvoir facilement supporter les "anciens" types d'appareil faisant partie du réseau;

- une présentation des alarmes et un système d'aide similaires à ceux utilisés par CAW de sorte qu'on puisse facilement les intégrer à l'environnement du CERN;

- la possibilité de supporter huit utilisateurs (c'est-à-dire huit SpectroGRAPHs) interactifs simultanément et au moins deux mille appareils. Cependant, nous avons pu remarquer un léger problème lorsque SPECTRUM était amené à gérer une telle quantité d'appareils. En effet, d'après les informations fournies par le logiciel, son utilisation du CPU dépassait largement les 100% (chose assez étonnante d'ailleurs) et le temps de réponse aux diverses opérations réalisées par un utilisateur était très élevé. De plus, les nombreux paquets de "polling" envoyés par SPECTRUM sur le réseau "engorgeaient" certains bridges qui ne savaient plus suivre, bloquant ainsi momentanément le réseau. Toutefois, nous sommes en mesure de nous demander si ce dernier problème est dû exclusivement à SPECTRUM, à la faible capacité de traitement des certains bridges, ou tout simplement aux deux en même temps;

- un support pour une large gamme de produits fabriqués par des vendeurs différents;

- la possibilité d'aller manipuler les informations contenues dans la base de données du logiciel pour permettre d'intégrer les applications externes telles que le "*Lego-Plot*" et "*AlarmMirror*";

- la possibilité de créer ses propres modules (grâce à l'"*External Protocol Interface*") pour pouvoir gérer des équipements ne supportant pas le protocole SNMP;

- la possibilité de configurer la génération des alarmes détectées par le SpectroSERVER et de la limiter aux appareils apparaissant dans les différentes vues;

- la présence d'une fonction d'aide pour expliquer la signification d'une alarme et ce que doit faire l'opérateur pour y remédier;

- une interface conviviale à l'utilisateur;

Parmi tous les logiciels de gestion de réseaux que le CERN a pu évaluer, SPECTRUM a été de loin le plus impressionnant. Il rencontre sans aucun doute tous les besoins du CERN, pas seulement au niveau des caractéristiques attendues et de sa grande flexibilité de configuration, mais aussi au niveau du soutien reçu de la part de la filiale anglaise durant tout le processus d'évaluation.

Le logiciel SPECTRUM est également le seul logiciel possédant un système d'alarme qui correspond aux attentes des opérateurs du réseau. Il effectue une analyse "intelligente" pour réduire le nombre de messages d'alarme apparaissant à l'écran. Ainsi, lorsqu'un appareil particulier occasionne un problème, seul un message concernant cet appareil est généré. Toutes les alarmes concernant les autres appareils subissant des problèmes consécutifs au premier problème ne sont pas prises en compte.

Ce sont tous ces avantages qui ont fait que le logiciel de gestion de réseaux SPECTRUM a été choisi pour gérer tout le réseau du CERN.
Mais SPECTRUM n'a pas que des avantages, il possède aussi des inconvénients. Nous avons déjà eu l'occasion d'en donner quelques-uns depuis le début de ce deuxième chapitre. Nous citerons encore d'autres inconvénients dans les chapitres 4, 5 et 6 consacrés à l'explication des projets que nous avons dû réaliser au CERN. Ainsi, nous montrerons entre autres les problèmes que le logiciel nous a posés et les solutions que nous avons choisies.

5. Dans quelle mesure SPECTRUM peut-il être considéré comme un système de gestion intégré ?

Dans cette partie, nous verrons de quelle manière le logiciel SPECTRUM peut être vu comme un logiciel de gestion intégré et quels sont les outils mis à la disposition des utilisateurs.
Mais au préalable, nous allons d'abord expliquer ce que nous entendons par un "système de gestion intégré".

5.1. Notion de "système de gestion intégré"

A l'heure actuelle, on voit de plus en plus apparaître des environnements intégrés dans lesquels un système de gestion principal (intégré) s'occupe de gérer tout un ensemble de systèmes de gestion de types différents. Ce système de gestion intégré est souvent appelé "Manager of Manager".
Chacun de ces systèmes de gestion s'occupe d'un réseau bien particulier disposant de ses propres agents et de son protocole de communication. Il peut aussi bien s'agir d'un réseau basé sur le monde TCP/IP, que d'un réseau basé sur le monde OSI, ou encore d'un simple réseau téléphonique.
Le système de gestion intégré offre en réalité un écran de contrôle virtuel unique par lequel l'opérateur surveille, contrôle et gère un grand nombre d'agents différents appartenant à des réseaux hétérogènes. Toutes les informations qui apparaissent à l'écran de contrôle du système de gestion intégré lui sont communiquées via le réseau par les différents systèmes de gestion.

Un système intégré se veut avant tout "transparent". En effet, lorsque l'utilisateur veut interagir avec un agent spécifique situé sur un réseau particulier, il ne doit pas s'occuper de connaître des informations telles que le système de gestion dont dépend l'agent, le protocole à utiliser, sa localisation physique, son adresse IP... C'est au système de gestion intégré de s'occuper de tous ces détails.

Le système de gestion intégré peut également gérer des agents directement sans passer par un système de gestion particulier.
Il pourrait ainsi gérer l'ascenseur d'un bâtiment en prévenant l'utilisateur de la survenance d'une panne et de sa cause probable, ainsi que la climatisation à l'intérieur d'un immeuble.

5.2. Où se situe SPECTRUM par rapport aux systèmes intégrés

En ce qui concerne le logiciel SPECTRUM, celui-ci peut s'adapter à plusieurs types d'environnements et communiquer avec un autre système de gestion grâce à son "Management Station Access Provider/External Protocol Interface" (MSAP/EPI).
En effet, bien que le logiciel SPECTRUM ait d'abord été développé comme un système de gestion de réseaux pour gérer des composants ou des applications d'un réseau informatique, il existe beaucoup d'autres domaines d'applications dans lesquels SPECTRUM peut être adapté. En tant que système de gestion, on peut le retrouver par exemple dans une industrie, dans un bureau comptable ou encore dans un système qui contrôle les stocks. SPECTRUM est également capable de gérer des logiciels tels qu'un tableur ou un gestionnaire d'une base de données externe.
Pour toutes ces applications qui ne sont pas courantes pour SPECTRUM, les développeurs doivent implémenter le MSAP/EPI. Il s'agit d'une interface générique qui est adaptable en fonction du protocole utilisé par un agent.

La figure 2.13. illustre la manière dont communiquent le logiciel SPECTRUM et le MSAP ("Management Station Access Provider") qui gère un quelconque autre appareil. Cette communication se réalise par l'intermédiaire du EPI ("External Protocol Interface").

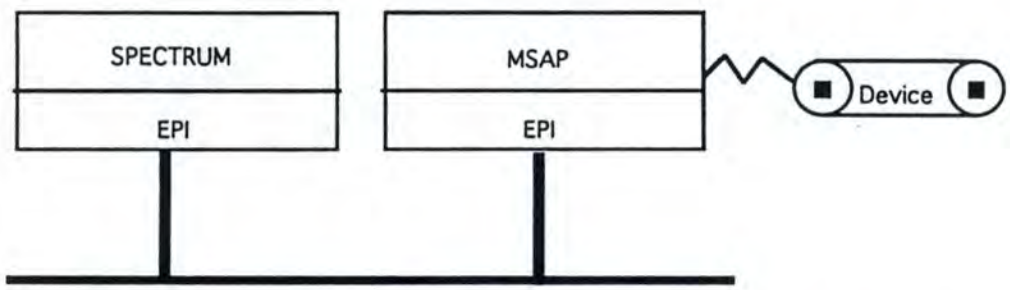


Figure 2.13. Communication entre SPECTRUM et le MSAP grâce à l'EPI.

Voici quelques exemples où il est nécessaire de faire appel au MSAP/EPI:

- l'application tourne dans un environnement différent de celui requis par SPECTRUM;
- les objets (équipements, logiciels d'application...) ont besoin d'être gérés par des protocoles de gestion propriétaires;
- le fabricant d'un équipement particulier souhaite que ce dernier soit tout de même géré par le SpectroSERVER malgré qu'il ne soit pas en possession d'un agent SNMP;
- un nombre important d'objets doivent être gérés (ou modélisés) et il est nécessaire de diminuer la charge du CPU du système de gestion pour améliorer ses performances.

5.2.1. Le "Management Station Access Provider"

Le concept du "Management Station Access Provider" (MSAP) a été développé pour étendre les capacités de gestion de SPECTRUM. Le MSAP permet à un objet (un composant du réseau, un robot dans une usine, un logiciel d'application...) d'avoir accès au système de gestion, le SpectroSERVER. Il peut tourner dans tout environnement: le même environnement que SPECTRUM, le même environnement que l'objet, ou tout autre environnement.
Le logiciel SPECTRUM communique avec le MSAP en utilisant TCP/IP sur une transmission fiable des données. Le MSAP peut utiliser n'importe quel mécanisme approprié pour communiquer avec l'objet qu'il doit gérer.

Le développement d'un MSAP fournit divers types de fonctionnalités:

- l'utilisateur peut créer, grâce au MSAP, une interface pour un équipement ne disposant pas d'un agent SNMP. En utilisant l'interface EPI, il est relativement facile de développer un MSAP capable de communiquer avec SPECTRUM et l'objet;

- les administrateurs qui ont acquis SPECTRUM et qui possèdent des équipements ne pouvant être gérés que par des protocoles de gestion propriétaires, ont la possibilité de développer un MSAP jouant le rôle d'intermédiaire entre SPECTRUM et les équipements en question. Le MSAP peut alors être vu comme un "traducteur" de requêtes;

- l'EPI peut tourner sur une plate-forme supportée par SPECTRUM avec des accès à TCP/IP alors que MSAP doit utiliser une plate-forme supportant le langage de programmation C. Le MSAP a été conçu pour pouvoir être utilisé dans un environnement en temps réel tel qu'une ligne d'assemblage d'une usine; d'un côté, il est capable de "parler" aux machines et de l'autre côté, il communique avec SPECTRUM à l'aide de TCP/IP;

- tout système hardware ou software est construit pour certaines spécifications et une lourde application peut pousser facilement un système à ses limites. En fonction de la plate-forme et du nombre d'objets gérés ou modélisés, le logiciel SPECTRUM pénalise les ressources du système telles que l'utilisation du CPU, l'espace en mémoire centrale, l'espace sur disque et les descripteurs de fichiers. En déplaçant une partie de la charge de SPECTRUM vers un MSAP, on peut dépasser les limites intrinsèques du CPU. De plus, si le MSAP tourne sur un système différent, on obtient des performances accrues;

A la vue de toutes ces fonctionnalités, nous pouvons dire que le MSAP fournit un mécanisme simple et facile à implémenter pour étendre les possibilités de SPECTRUM. Il joue en quelque sorte le rôle d'un "proxy-agent".

5.2.2. Le module de gestion de l'"External Protocol Interface"

Pour que le logiciel SPECTRUM puisse représenter des objets ou des équipements gérés à l'aide du MSAP/EPI, l'utilisateur doit acheter le module de gestion de l'EPI. Il comprend deux types de modèles qui constituent des points de dérivation pour créer de nouveaux types de modèle représentant le MSAP et l'équipement qu'il gère. Si l'on désire en ajouter d'autres, il suffit d'utiliser l'éditeur de types de modèles (MTE). Si l'on éprouve également le besoin d'ajouter de l'intelligence pour disposer d'un emploi plus précis de l'information en provenance de l'équipement et du MSAP/EPI, l'utilisateur a la possibilité d'écrire de nouvelles règles d'inférence.

5.2.3. L'"External Protocol Interface"

Le logiciel SPECTRUM et le MSAP communiquent entre eux avec une interface bien définie appelée l'"External Protocol Interface". L'EPI s'occupe de cacher les détails de communication entre SPECTRUM et le MSAP. Lorsque le MSAP a un lien avec l'EPI, il dispose d'un accès à tous les services de communication fournis par l'EPI et les applications sous-jacentes.

Il existe deux scénarii qui illustrent la communication entre SPECTRUM et le MSAP. Le premier consiste en un scénario de "question/réponse" initialisé par SPECTRUM afin d'échanger des informations de gestion avec le MSAP. Le second scénario, moins fréquent, se déroule lorsque le MSAP (habituellement sur demande de l'objet géré) a besoin d'envoyer des messages non sollicités vers SPECTRUM. Cela s'apparente au mécanisme de trappe.

6. Conclusion

En guise de conclusion, nous pouvons dire que le logiciel SPECTRUM est un système de gestion de réseaux flexible et très puissant [INT94]. Il recouvre bien les cinq domaines que doit couvrir tout bon système de gestion. Pour rappel, ces cinq domaines ont été décrits au début de ce chapitre dans le point 2. De plus, il fournit quelques fonctionnalités attrayantes, telles que la capacité de déterminer des dépendances entre des composants du réseau ou de supprimer de "fausses" alarmes.
Le logiciel SPECTRUM peut également être considéré comme un système de gestion intégré au regard des capacités offertes par le "Management Station Access Provider" et l'"External Protocol Interface". En effet, grâce au MSAP et à l'EPI, l'utilisateur est capable d'implémenter une interface générique spécifique de telle sorte que SPECTRUM puisse gérer un agent particulier n'ayant aucun rapport avec un composant d'un réseau informatique.

Mais comme le logiciel SPECTRUM se veut flexible, il est aussi complexe. La formation des utilisateurs est donc très importante; sans elle, l'utilisateur pourrait éprouver bon nombre de difficultés.

Chapitre 3:
Notre environnement de travail

Ce chapitre servira en quelque sorte d'introduction aux trois chapitres suivants. Nous allons d'abord définir l'environnement dans lequel nous avons développé nos trois projets. Ensuite, avant de détailler en profondeur chacun de ces trois projets dans les chapitres 4, 5 et 6, nous allons les présenter brièvement et introduire quelques concepts utiles à la compréhension ultérieure de ce mémoire. Nous expliquerons également deux applications spécifiques qui se rapportent directement à nos deux premiers projets, le "*Lego-Plot*" et "*AlarmMirror*".

1. Le contexte dans lequel nous avons développé nos projets

Dans le chapitre précédent, nous avons décrit les grandes caractéristiques du logiciel de gestion de réseaux SPECTRUM et nous avons pu constater que la gestion d'un réseau n'était pas une mince affaire surtout lorsque ce dernier était de grande ampleur.

Jusqu'à présent, le logiciel SPECTRUM n'est pas encore en production au CERN et se trouve toujours dans sa phase de développement. Cette phase de développement est entièrement consacrée, comme son nom l'indique, au développement du logiciel. Pendant toute la période que dure cette phase importante, quelques personnes sont chargées de comprendre le logiciel, de l'essayer, de le configurer et de l'adapter aux structures existantes. En d'autres termes, cela signifie que tant que la phase de développement de SPECTRUM ne sera pas terminée, il ne sera pas utilisé par les différents administrateurs et opérateurs du réseau. Pour le moment, ces différents administrateurs et opérateurs se basent toujours sur le logiciel de gestion de réseaux actuel CAW

qui, lui, est en production depuis plusieurs années. Le logiciel CAW a été développé par le CERN lui-même afin de disposer d'un système de gestion de réseaux ad hoc. Il constitue un ensemble d'utilitaires pour contrôler et gérer l'ensemble du réseau du CERN.

Mais pour des raisons financières, le responsable du groupe CS ("Communications Systems") a décidé récemment de remplacer le logiciel CAW par le logiciel commercial SPECTRUM. Pour que cette transition occasionne le moins de perturbations possible au sein du travail de gestion du réseau, le logiciel SPECTRUM tourne actuellement sur une machine particulière parallèlement au logiciel CAW. Ceci permet aux quelques personnes chargées du développement de SPECTRUM de "jouer" avec le logiciel et ainsi d'observer ses divers comportements en fonction des actions qu'elles entreprennent, sans pour cela gêner la gestion réelle du réseau par CAW.

Durant toute la durée de notre stage au CERN, nous avons participé à la phase de développement du logiciel SPECTRUM. Nous avons travaillé de concert avec quelques personnes pour essayer d'intégrer le nouveau logiciel de gestion de réseaux dans la structure existante du CERN.
En effet, à côté du logiciel principal CAW, on trouve deux grandes applications qui manipulent des données fournies par ce logiciel. Il s'agit de l'application "*Lego-Plot*" et de l'application "*AlarmMirror*". La première fournit une représentation graphique sur les performances des différents segments constituant le réseau pour tenter d'"anticiper" les problèmes réels qui pourraient survenir à leur niveau (risque de congestion, taux d'erreur trop important...). La seconde informe les opérateurs de toutes les alarmes qui ont été détectées sur le réseau et de leur cause probable pour que ces derniers puissent y répondre le plus rapidement possible.

Nos deux premiers projets ont d'abord consisté à rechercher des méthodes par lesquelles on pouvait extraire de la base de données du logiciel SPECTRUM les informations requises par ces deux applications. Ensuite, pour chacune de ces deux applications, nous avons dû créer un programme qui, recevant en entrée les données extraites, était chargé de les présenter aux applications.

Notre troisième et dernier projet était un peu différent des deux premiers. Il se rapprochait plutôt de la "configuration graphique" du logiciel. En effet, nous avons dû écrire un programme qui s'occupait de copier les icônes situées dans la vue "*Topologie*" et représentant les équipements du réseau du CERN (bridges, routeurs, pingables...) dans la vue "*Localisation*" correspondant à l'emplacement physique en termes de bâtiments et/ou de salles où se trouvent réellement ces équipements.

2. L'application "Lego-Plot"

Pour essayer de prévenir les problèmes qui pourraient surgir sur l'ensemble de son réseau, le CERN a développé, à côté du logiciel CAW, une application appelée "*Lego-Plot*". Il s'agit d'une application de statistiques qui s'occupe de représenter graphiquement les performances de tous les

segments du réseau par tranche d'heure pour une journée (taux d'erreur, nombre de paquets par seconde, nombre de paquets émis en multi-broadcast...).

L'application "*Lego-Plot*" réalise ces différentes représentations graphiques à partir des données recueillies auprès des différents segments appartenant au réseau et enregistrées dans la base de données du logiciel CAW [GAM89]. Pour obtenir les performances d'un segment particulier du réseau, comme par exemple le nombre de paquets qui y transitent par seconde, il suffit d'aller interroger une machine connectée sur ce segment par le mécanisme de "polling" en lui demandant le nombre de paquets qu'elle voit effectivement passer par seconde.
En principe, une seule machine suffit puisque, théoriquement, toutes les autres machines connectées sur ce même segment devraient fournir les mêmes résultats. Mais en pratique, c'est différent; comme il se peut que la machine qui fasse l'objet d'un "polling" ne fonctionne pas parfaitement et fournisse des résultats incorrects, on va réaliser un "polling" de toutes les machines connectées sur le segment. De cette manière, si toutes les machines donnent comme réponse le même résultat, on considère ce résultat correct. Si par contre, une (ou plusieurs) machine donne des résultats différents, on prend comme résultat correct celui donné par la majorité des machines. On peut, par conséquent, également supposer que la ou les machines ayant fourni les résultats différents sont sujettes à des problèmes et donc en informer les opérateurs.

Les représentations graphiques données par l'application "*Lego-Plot*" sont des représentations sous la forme d'histogramme en trois dimensions (voir Annexe 2). On y trouve:

- un taux à visualiser en ordonnée verticale (par exemple, le nombre de paquets qui transitent par seconde);
- le nom du segment;
- l'heure de la journée à laquelle correspond le taux.

Si l'utilisateur le souhaite, il peut visionner les graphiques selon plusieurs angles de vues (de face, de derrière, de côté...) et en couleurs. Pour éviter une surcharge des graphiques, les segments possédant un taux inférieur à 10^{-4} unité par seconde ne sont pas pris en compte.

Actuellement, les données à partir desquelles l'application "*Lego-Plot*" réalise les graphiques des performances des différents segments du réseau sont lues dans la base de données du logiciel CAW grâce à un programme développé dans le langage Fortran. Ce programme écrit ensuite ces données dans un fichier selon un format compréhensible par l'application.

Comme le "*Lego-Plot*" utilise une représentation graphique qui donne à l'opérateur une bonne perception intuitive des performances du réseau, le CERN désire que l'application soit toujours opérationnelle lorsque le logiciel SPECTRUM sera amené à suppléer le logiciel CAW. C'est pourquoi il est nécessaire d'adapter l'application au nouvel environnement qu'offre le logiciel SPECTRUM.

Notre premier projet consiste donc à créer un programme en langage C qui se charge d'abord d'extraire les informations nécessaires à l'application "*Lego-Plot*" de la base de données de SPECTRUM. Ensuite, tout comme avec le logiciel CAW, le programme doit écrire ces données suivant un format bien défini dans un fichier qui sera fourni en entrée à l'application "*Lego-Plot*".

3. Le "système d'alarme" du CERN

3.1. Introduction

Dans un réseau à grande échelle, il peut arriver qu'un problème survienne au niveau d'un de ses composants tel qu'un bridge, un routeur, un "pingable",... Ces alarmes peuvent avoir trait à la perte de contact avec un équipement, à un risque de congestion d'un équipement, à un taux d'erreur trop important pour un équipement, etc...
Il s'agit alors de les détecter au plus vite pour y remédier.
Pour le moment, c'est le logiciel CAW qui est chargé, entre autres, de détecter toutes ces alarmes. A côté de cela, il dispose d'une application "*AlarmMirror*" qui permet d'afficher sur l'écran de l'opérateur toutes les informations sur une alarme provenant d'un équipement du réseau.

3.2. Qu'est-ce qu'un "système d'alarme"

Un "système d'alarme" consiste d'une part, en un ou plusieurs programmes de contrôle qui surveillent l'ensemble des équipements du réseau et, d'autre part, en un système qui se charge de l'affichage des alarmes sur l'écran d'un opérateur [GAM88]. Il tente de présenter, sur un simple et même écran, tous les messages indiquant qu'une intervention humaine est requise pour une alarme. Pour le moment au CERN, le système d'alarme est constitué par le logiciel CAW et l'application "*AlarmMirror*". CAW joue le rôle du programme de contrôle et "*AlarmMirror*" tient celui du système qui se charge de l'affichage des alarmes sur l'écran.

Un "système d'alarme" ne doit pas remplacer les applications qui représentent visuellement l'état des équipements, les lignes de communication ou les stations de travail (pour rappel, avec le logiciel SPECTRUM, ce rôle est joué par le SpectroGRAPH). De telles applications sont toujours utiles pour rassembler des informations plus détaillées à propos d'un problème indiqué par un message d'alarme.
Elles sont également chargées d'indiquer au système d'affichage toute occurrence de problèmes pour qu'il puisse les présenter sur l'écran de l'opérateur.

Les messages d'alarme restent affichés sur l'écran de l'opérateur aussi longtemps que le programme de contrôle n'a pas détecté que le problème a été résolu ou que l'opérateur ne les a pas effacés manuellement. Tout nouveau message apparaît en haut de l'écran de façon à ce que

l'opérateur puisse facilement discerner les derniers problèmes qui viennent de survenir sur le réseau.

On peut trouver également une information qui procure une aide interactive pour "rappeler" à l'opérateur ce que veut dire tel ou tel message d'alarme, spécialement dans le cas où un certain message d'alarme n'a plus été affiché depuis plusieurs mois.
Cette information d'aide s'occupe aussi de fournir les coordonnées (nom, prénom, numéro de téléphone...) des personnes à contacter en fonction du type de problème.

Un "système d'alarme" doit, pour bien faire, garder un fichier reprenant les informations sur les moments de déclenchement et d'annulation des alarmes ainsi que tous les messages de "login" sur le système. Ceci permet de réaliser ultérieurement une analyse de la séquence des événements ou encore des statistiques sur la fiabilité des équipements.

Dans le cas particulier du CERN, le "système d'alarme" est auto-vérificateur. En effet, il vérifie que le programme de contrôle qui devrait tourner, à savoir CAW, tournent réellement. Pour ce faire, tout programme de contrôle a la possibilité d'informer le "système d'alarme" de la fréquence à laquelle il doit être déclenché et de la période de temps qu'il occupe. De cette façon, le "système d'alarme" peut vérifier si le programme de contrôle se déclenche bien à la fréquence prévue et ne dépasse pas la période de temps qui lui est allouée.
Si le programme de contrôle n'est pas déclenché à la dite fréquence, le "système d'alarme" génère un message d'alarme en indiquant le temps depuis lequel le programme de contrôle n'a plus été activé.
Ce qu'on peut toutefois regretter, c'est que le "système d'alarme" ne s'occupe pas de vérifier si le système d'affichage à l'écran fonctionne correctement.

3.3. Format des messages transmis au système d'affichage

L'ensemble des messages d'alarme et des messages de "login" transmis au système d'affichage par le programme de contrôle sont présentés sous une forme textuelle. Ils ont le format suivant:

<code>:<système><heure>:<message ID>:<texte>

Le <code> peut prendre plusieurs valeurs:

- AL : pour signaler un message d'alarme;
- WA : pour signaler un message d'avertissement ("Warning") dans le cas, par exemple, d'un risque de congestion;
- LO : pour signaler un message de "login";

- CA : pour annuler ("Cancel") un message d'alarme. En fonction de l'identifiant du message contenu dans <message ID> le système d'affichage connaît le message d'alarme qu'il convient d'effacer;

- RI : pour spécifier l'intervalle de temps qui existe entre deux exécutions du programme de contrôle. Le <message ID> représente l'intervalle en secondes. Le <texte> fournit de l'information supplémentaire à l'opérateur telle que la machine sur laquelle tourne le programme, sa version...

- C* : pour annuler tous les messages d'alarme et d'avertissement pour un programme de contrôle donné. Ce message est envoyé lorsqu'on redémarre le programme de contrôle.

Le <système> représente le nom du programme de contrôle.
L' <heure> n'est pas obligatoire. Cependant, le programme de contrôle peut spécifier, s'il le souhaite, l'heure à laquelle le message a été généré.
Le <message ID> représente l'identifiant d'une alarme. De cette manière, lorsque que le système d'affichage reçoit un nouveau message sur une alarme avec un identifiant particulier, il peut savoir s'il s'agit d'une alarme qui est déjà affichée ou non à l'écran. Dans l'affirmative, il convient de remplacer l'ancien message, qui possède le même identifiant d'alarme, par le nouveau message. Dans la négative, il suffit d'afficher à l'écran le nouveau message d'alarme. Ainsi, un <message ID> apparaîtra au plus une seule fois sur l'écran d'affichage.
Quant au <texte>, il constitue simplement une brève description de la cause de l'alarme.

Voici un exemple de message:

AL:SPECTRUM:235: <nom d'un équipement> Contact lost

3.4. Description de l'écran d'affichage

L'écran d'affichage principal est divisé en deux zones: la partie supérieure est réservée aux messages indiquant des alarmes, tandis que la partie inférieure traite des messages de "login".

Chaque message est présenté à l'écran dans le format suivant:

<heure> <date> <système> <message>

L'<heure> et la <date> sont ajoutées par le système d'affichage des alarmes en fonction du moment où celles-ci lui parviennent. Le <système> est un nom d'une longueur de six caractères qui indique le programme de contrôle qui produit le message. Et le <message> constitue un texte en rapport avec une alarme, un avertissement ou un "login".

Toutes les nouvelles alarmes sont affichées avec un clignotement à l'écran et provoquent un petit avertissement sonore. Une fois que l'opérateur en a pris connaissance en allant cliquer dessus, le clignotement de la nouvelle alarme s'arrête ainsi que l'avertissement sonore s'y rapportant.

Les alarmes seront normalement effacées par le programme de contrôle qui les enlèvera de l'écran lorsqu'il recevra un message indiquant leur annulation. Cependant, les alarmes peuvent également être effacées "manuellement" à l'aide de la souris et des touches de fonction.
Les alarmes annulées par le programme de contrôle sont d'abord maintenues à l'écran pendant trente secondes avec, comme couleur de fond, le vert. Ensuite seulement, elles disparaissent. Ceci permet à l'opérateur d'avoir le temps de voir une alarme même si elle est de suite annulée. Les arrivées répétées d'alarmes déjà présentes à l'écran auront pour seul effet de mettre à jour l'information du message des alarmes; le temps et la date ne changeront pas et il n'y aura ni clignotement, ni avertissement sonore. De plus, si l'alarme a un "fond vert", elle reprend l'apparence d'une alarme "normale". Cependant, si l'alarme n'est plus présente à l'écran, elle sera considérée comme une nouvelle alarme lors de sa réapparition.

Les avertissements, en rapport avec des alarmes d'un niveau mineur, sont traités pratiquement de la même manière que les alarmes d'un niveau majeur. La seule différence se situe dans l'avertissement sonore qui ne retentit que durant une période de cinq secondes.

L'opérateur peut obtenir une aide sur tout message affiché concernant une alarme en le sélectionnant à l'aide de la souris et en utilisant la touche de fonction pour l'aide. La disponibilité d'une aide en information dépend du contenu du fichier d'aide. Si aucune alarme n'a été sélectionnée (le curseur se situe dans la partie de l'écran réservée aux messages de "login"), l'information qui est présentée est celle en rapport avec le système d'affichage lui-même.

Afin de garder facilement disponible autant d'informations pertinentes que possible, les programmes de contrôle générant trop de messages d'alarme (plus que trois) sont représentés par un simple message de la forme:

<heure> <date> ALARM Several Alarms for system=<program> Class=<class>

Le <program> est le nom du programme de contrôle qui a généré le message et la <class> représente une sous-classe de messages pouvant être générés par le programme de contrôle.

Les gens intéressés par les messages d'alarme en provenance du réseau peuvent aussi demander un "miroir" de l'écran d'affichage principal. Cela leur permet de visionner les messages d'alarme à partir de leur station de travail ou de leur terminal. Tous les messages d'alarmes ou de "login" qui arrivent sur le système d'affichage principal sont copiés vers ces écrans secondaires. Lorsqu'un utilisateur ouvre un "miroir", tous les messages d'alarmes et les informations qui se trouvent sur l'écran principal y sont envoyés.

Une certaine touche de fonction sur le système d'affichage principal permet de connaître la liste des stations de travail qui ont ouvert un "miroir" ainsi que le nom de l'utilisateur qui l'a demandé.

Si la communication avec le système central est perdue, le "miroir" tentera de rétablir le contact toutes les vingt secondes. Durant ce temps, le message "NO Communication" est affiché et à chaque fois que le "miroir" essaye de rétablir le contact en vain, le message "NO ALARMS" apparaît.

3.5. L'application "*AlarmMirror*"

Au CERN, nous avons vu que l'application qui joue le rôle du système d'affichage s'appelle "*AlarmMirror*". Cette application s'occupe d'afficher sur l'écran de l'opérateur tous les messages se rapportant à des alarmes détectées sur le réseau. Elle reçoit tous ces messages grâce à un programme de contrôle faisant partie intégrante du logiciel CAW. A chaque occurrence d'une nouvelle alarme sur le réseau ou lorsqu'une alarme existante change de statut, le programme de contrôle se charge de transmettre au système d'affichage un message décrivant l'alarme grâce au protocole UDP (User Datagram Protocol) et à l'utilisation de mécanismes permettant la communication entre applications (les "sockets").

Le logiciel SPECTRUM possède également son propre module qui se charge de détecter les alarmes survenant sur le réseau, de les enregistrer dans sa base de données et de les afficher dans une fenêtre prévue à cet effet: la vue "*Alarme*" (voir figure 2.8.) ([SPEC5] & [SPEC6]). Dans cette vue, les alarmes ainsi que les date et heure auxquelles elles ont été détectées sont affichées avec des couleurs différentes en fonction de leur gravité: gris, jaune, orange ou rouge. L'opérateur peut obtenir plus de détails sur une alarme, tels que sa cause probable, comment y remédier, quelle personne contacter... en allant cliquer dessus. De plus, si la vue "*Alarme*" est ouverte (ou icônisée) lors de la survenance d'une alarme ou d'un changement de statut d'une alarme existante, un avertissement sonore retentit.

Comme le CERN désire également garder opérationnelle son application "*AlarmMirror*", notre deuxième projet consiste à aller rechercher dans la base de données du nouveau logiciel SPECTRUM les informations sur toutes les alarmes (heure et date auxquelles elles se sont produites, niveau de sévérité, causes probables...) et à les transmettre à l'application.
La vue "*Alarme*" de SPECTRUM reste cependant totalement opérationnelle pour quiconque voulant l'utiliser.

4. Le projet de configuration des vues "Localisation"

Dans cette partie, nous allons présenter les grandes lignes de notre troisième projet. Mais nous aimerions attirer l'attention sur le fait que ce troisième projet n'a pas de rapports directs avec nos deux premiers projets qui avaient pour but d'adapter SPECTRUM à son nouvel environnement. Il n'utilise pas non plus des applications externes comme le "*Lego-Plot*" ou "*AlarmMirror*".
Pour réaliser cette présentation, nous nous sommes inspirés de deux tomes [SPEC5] et [SPEC6] proposés par la société Cabletron de SPECTRUM qui nous ont servi à décrire en détails les vues "*Localisation*" entrant fortement en ligne de compte dans ce troisième projet.

4.1. Le caractère hiérarchique des vues "*Localisation*"

Dans le deuxième chapitre, nous avons vu que les vues "*Localisation*" représentent les équipements du réseau en termes d'emplacements physiques. On peut créer plusieurs vues de ce type pour construire une hiérarchie de vues "*Localisation*".
De cette façon, on arrive à situer le réseau d'une manière de plus en plus précise. Le niveau le plus haut que l'on puisse obtenir dans cette hiérarchie est sa localisation mondiale. Ensuite, en allant cliquer sur le pays dans lequel il est établi, une nouvelle vue "*Localisation*" s'ouvre sur le pays en question avec ses villes et ses agglomérations. Et ainsi de suite, en procédant de la même manière, on descend de plus en plus dans la hiérarchie. On peut alors obtenir d'une manière plus détaillée, la ville et le secteur dans lesquels le réseau se trouve, les bâtiments et les salles que ses divers composants occupent (voir figure 2.7.).

Voici la hiérarchie complète qu'offre le logiciel SPECTRUM pour représenter la localisation de son réseau:

- **Monde** : il constitue le niveau le plus haut dans la hiérarchie;
- **Pays** : il est employé pour représenter une nation;
- **Région** : elle représente une grande zone à l'intérieur d'un pays;
- **Site** : il correspond à une localisation plus petite telle qu'un établissement militaire, un campus universitaire ou un parc industriel;
- **Secteur** : il constitue une partition d'un site. Par exemple, on peut diviser un parc industriel en plusieurs secteurs;

- **Bâtiment** : il s'agit d'une simple structure physique;
- **Etage** : il constitue un moyen pour effectuer une partition d'un immeuble composé de plusieurs étages;
- **Section** : un étage peut être divisé en plusieurs sections;
- **Salle** : une salle correspond au premier niveau de localisation dans lequel on peut représenter le modèle d'un équipement;
- **Compartiment** : il constitue le niveau le plus bas dans la hiérarchie et sert à faciliter l'organisation des modèles dans les salles qui en contiennent un grand nombre.

Cependant, lorsqu'un administrateur représente un réseau à l'aide des vues "*Localisation*", il n'est pas obligé de passer par tous les niveaux qui constituent la hiérarchie globale.

4.2. Le projet

Notre troisième projet se rapporte aux emplacements physiques des divers équipements figurant dans les vues "*Topologie*" de SPECTRUM (bridges, routeurs et "pingables") et situés dans les différents bâtiments du CERN. Il peut être divisé en deux parties.

4.2.1. Première partie

La première consiste à assigner une valeur à l'attribut "*Location*" apparaissant dans la vue "*Information*" d'un équipement. De sorte que, lorsque l'utilisateur ouvre une vue "*Information*" sur un équipement particulier, il puisse connaître son emplacement physique en termes de bâtiment et de salle.

4.2.2. Seconde partie

La seconde partie se situe au niveau des vues "*Localisation*" représentant des bâtiments ou des salles. Il convient d'aller copier dans ces différentes vues les icônes représentant les équipements que l'on trouve dans les vues "*Topologie*". La vue "*Localisation*" dans laquelle on copie une icône dépend de l'emplacement physique de l'équipement qu'elle représente. De cette manière, chaque utilisateur a la possibilité de connaître tous les équipements situés dans un bâtiment ou dans une salle. Pour cela, il lui suffit d'ouvrir la vue "*Localisation*" se rapportant au bâtiment ou à la salle.

4.3. La base de données ORACLE

Pour trouver l'endroit où est situé un équipement particulier, il existe une base de données ORACLE. Celle-ci reprend, entre autres, la localisation de la plupart des équipements composant le réseau en termes de bâtiment et, dans beaucoup de cas, de salle dans lesquels se trouve l'équipement. Les bâtiments sont identifiés par des numéros tandis que les salles sont identifiées soit par un étage et un numéro, soit par un nom.

4.3.1. L'identifiant d' un équipement

Pour obtenir l'emplacement physique d'un équipement enregistré dans la base de données de SPECTRUM, il faut aller interroger la base de données ORACLE au moyen d'une requête SQL contenant une clé d'accès qui identifie l'équipement au sein de tous les autres équipements. Dans le cadre de notre travail, nous utiliserons l'adresse physique d'un équipement comme identifiant.

4.3.2. La syntaxe de la localisation

Normalement, la localisation d'un équipement doit posséder la syntaxe suivante:

<n° du bâtiment> <n° d'étage - n° de la salle> <:nom de la salle>

Le <n° du bâtiment> et le <n° d'étage - n° de la salle> sont obligatoires tandis que le <:nom de la salle> est optionnel. Mais quand ce dernier est présent, il prévaut sur le <n° d'étage - n° de la salle>. Pour mieux comprendre, prenons deux exemples:

- "Localisation: 513 2-050" signifie que l'équipement se trouve dans le bâtiment n° 513, au 2ème étage dans la salle n° 050;

- "Localisation: 15 1-024 : NETWORKS" signifie que l'équipement se trouve dans le bâtiment n° 15, dans la salle dont le nom est "NETWORKS". En fait, lorsqu'on a un nom de salle, on ignore tout simplement le numéro de la salle et l'étage auquel elle se trouve.

4.3.3. Vérification de la syntaxe de la localisation

On nous a également demandé de procéder à une vérification de la syntaxe de la "proposition" décrivant la localisation d'un équipement, après l'avoir trouvée dans la base de données ORACLE. Si cette syntaxe n'est pas conforme à celle décrite au point précédent, on ne place pas l'équipement dans la vue "*Localisation*" adéquate et la valeur de l'attribut "*Location*" de la vue "*Information*" de l'équipement n'est pas assignée. Par contre, l'identifiant de l'équipement

ainsi que sa localisation sont écrits dans un fichier reprenant tous les problèmes rencontrés. Ce fichier sera ensuite transmis au gestionnaire de la base de données ORACLE pour qu'il puisse corriger la syntaxe de toutes ces localisations possédant une syntaxe non conforme.

Si, par contre, la syntaxe de la phrase décrivant la localisation de l'équipement est correcte, on assigne la valeur de l'attribut "*Location*" apparaissant dans la vue "*Information*" de l'équipement en question avec la localisation trouvée dans la base de données ORACLE.

Ensuite, si le bâtiment et la salle existent dans la base de données de SPECTRUM, on copie l'icône représentant l'équipement de la vue "*Topologie*" vers la vue "*Localisation*" de la salle adéquate.
Si, par contre, seul le bâtiment existe, on copie l'icône représentant l'équipement dans la vue "*Localisation*" qui correspond au bon bâtiment en attendant que la salle soit créée. A côté de cela, on écrit dans un fichier à l'attention de l'administrateur de SPECTRUM le numéro ou le nom de la salle à créer ainsi que le numéro du bâtiment dans lequel elle doit être créée.

Et enfin, si le bâtiment en question n'existe pas non plus, on écrit seulement le numéro du bâtiment à créer dans le fichier à l'attention de l'administrateur de SPECTRUM.

5. Conclusion

Dans ce chapitre, nous nous sommes contentés de poser les bases de nos travaux réalisés au CERN durant toute la durée de notre stage. Nous avons tout d'abord décrit brièvement le logiciel actuel de gestion de réseaux ainsi que les deux grandes applications qui s'y greffent et qui ont été développées dans le but de fournir une meilleure perception des performances du réseau du CERN. Ensuite, nous avons donné les spécifications générales de chacun de nos trois projets.

Les trois chapitres qui vont suivre maintenant auront chacun trait à l'un de nos projets. Ils expliqueront les problèmes rencontrés lors de leur conception, la manière dont ils ont été conçus, ainsi que l'architecture adoptée pour la solution.

Pour terminer ce chapitre et avant d'approfondir nos trois projets avec leurs programmes, nous vous proposons à la page suivante la figure 3.1. qui donne une illustration de toutes les relations existant entre nos différents programmes et les composants du logiciel SPECTRUM. Cette figure est là pour faciliter la compréhension de l'enchaînement de tous les programmes et pour mieux visualiser l'environnement dans lequel ils doivent tourner.

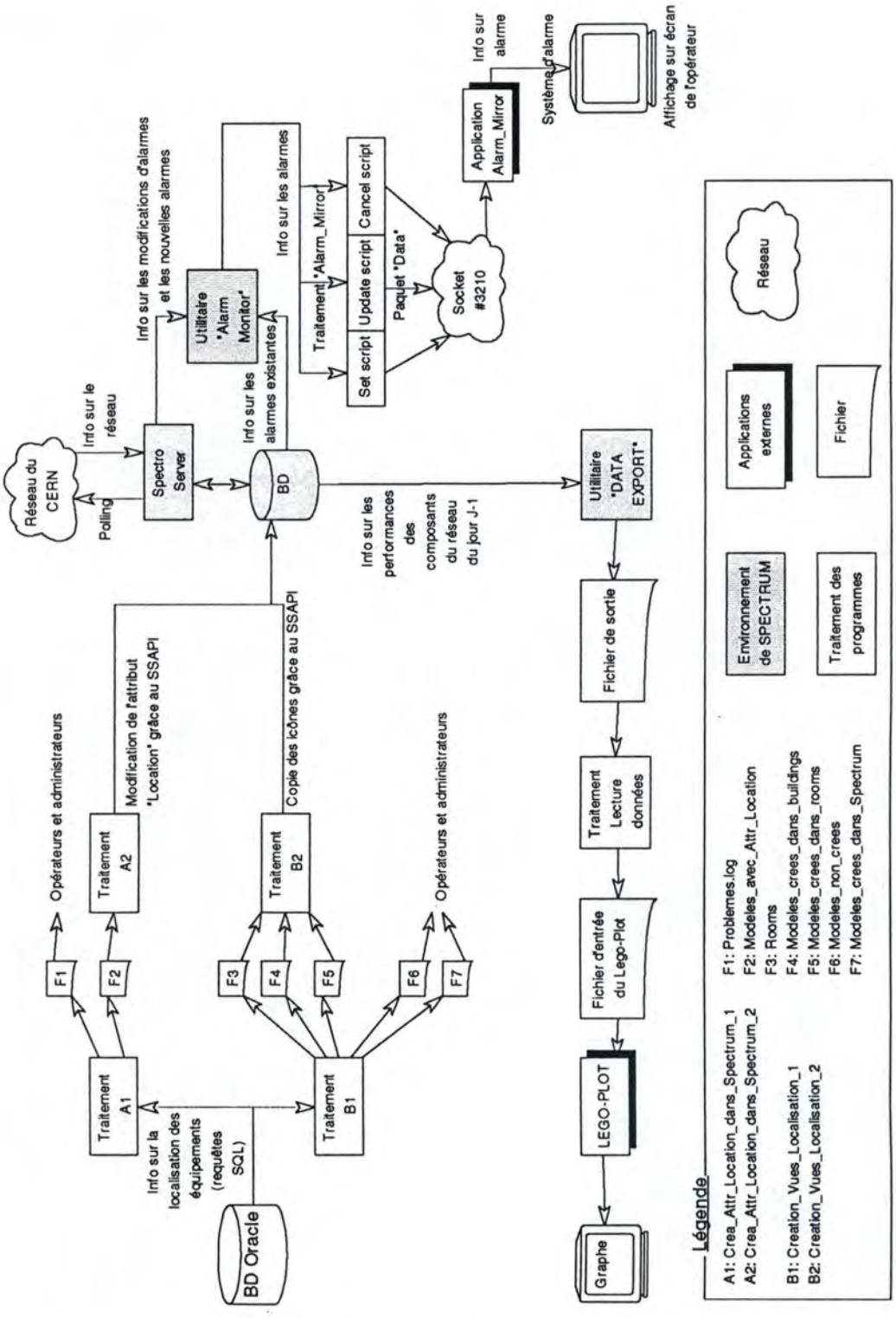


Figure 3.1. Relations entre SPECTRUM et nos trois projets.

Chapitre 4:
L'application "*Lego-Plot*"

Dans ce quatrième chapitre, nous allons présenter le premier projet qu'il nous a été demandé de réaliser, le programme "LectureDonnees".
Nous donnerons d'abord la définition précise du problème qui nous était posé. Ensuite nous expliquerons une fonction du logiciel SPECTRUM, la fonction "DATAEXPORT", qui nous a été bien utile pour la réalisation du projet. Nous verrons également le format spécial du fichier demandé par l'application "*Lego-Plot*". Et finalement, nous montrerons la façon dont a été conçu le programme, les problèmes rencontrés et les solutions adoptées.

1. Définition du problème

Le logiciel SPECTRUM dispose d'une base de données qui comprend, entre autres, tout un ensemble d'informations relatives aux performances des différents équipements du réseau. Ces informations, qui sont obtenues par le mécanisme de "polling" dont nous avons parlé au point 3.3.2. du deuxième chapitre, peuvent être importées dans diverses applications. En voici quelques exemples:

- un générateur de rapport;
- une base de données extérieure;
- une feuille de calcul;

- une application graphique qui élabore des statistiques sur les performances du réseau.

Dans notre cas, nous nous trouvons devant une application graphique qui s'appelle le "*Lego-Plot*".

On nous a demandé d'élaborer un programme que nous avons appelé "LectureDonnees" qui doit jouer le rôle d'une interface entre le nouveau logiciel SPECTRUM et l'application "*Lego-Plot*". Cette interface doit permettre de transférer de la base de données du logiciel les informations dont a besoin le "*Lego-Plot*" pour réaliser ses différents graphiques.

Mais avant de réfléchir à la façon de concevoir un tel programme, il nous est indispensable de trouver une méthode pour extraire des informations de la base de données de SPECTRUM.

Ensuite seulement, nous pouvons commencer à réfléchir sur la manière avec laquelle les informations extraites seront analysées par le programme et écrites dans un fichier baptisé "fichier d'entrée du *Lego-Plot*". Les informations écrites dans ce fichier doivent disposer d'un format bien défini afin qu'elles soient compréhensibles par le "*Lego-Plot*". Une fois ce fichier créé, il sera fourni en entrée à l'application pour qu'elle dessine les graphiques représentant les performances des différents segments du réseau.

2. La fonction "DATAEXPORT" du logiciel SPECTRUM

La fonction "DATAEXPORT" proposée par le logiciel SPECTRUM permet à l'utilisateur d'exporter des informations contenues dans sa base de données. Cette fonction convertit les informations exportées dans un format ASCII et les écrit dans un fichier baptisé "fichier de sortie" de manière telle que ces informations puissent être lues et utilisées par une quelconque application.

Il existe deux aspects à l'exportation des données avec la fonction "DATAEXPORT":

1. La création d'un "fichier de définition" qui spécifie quelles données doivent être exportées;

2. L'exécution à proprement parler utilisant les critères spécifiés dans le "fichier de définition".

2.1. Le "fichier de définition"

Pour se servir de cette fonction "DATAEXPORT", l'utilisateur doit d'abord définir un "fichier de définition" qui spécifie plusieurs points:

2.1.1. Le(s) type(s) de données à exporter

Nous pouvons prendre comme exemples de type de données à exporter:

- un type de modèle particulier: l'ensemble des bridges, des routeurs, des "pingables", des interfaces...
- un attribut contenu dans la MIB pour tout modèle qui en dispose;
- une relation entre des modèles: "Est connecté à", "Est parent de", "Est enfant de"...;
- des statistiques sur des équipements;
- l'occurrence d'événements extraordinaires.

Dans notre projet, nous nous occuperons seulement des interfaces qui sont appelées dans SPECTRUM, des "Gen_IF_Port".

2.1.2. Le nom et le répertoire du "fichier de sortie"

L'utilisateur doit donner le nom qu'il assigne au "fichier de sortie" dans lequel seront écrites les informations ainsi que le chemin d'accès à ce fichier.

La figure 4.1. illustre la fenêtre par laquelle l'utilisateur définit le nom et le répertoire du "fichier de sortie" ainsi que le(s) type(s) de données à exporter.

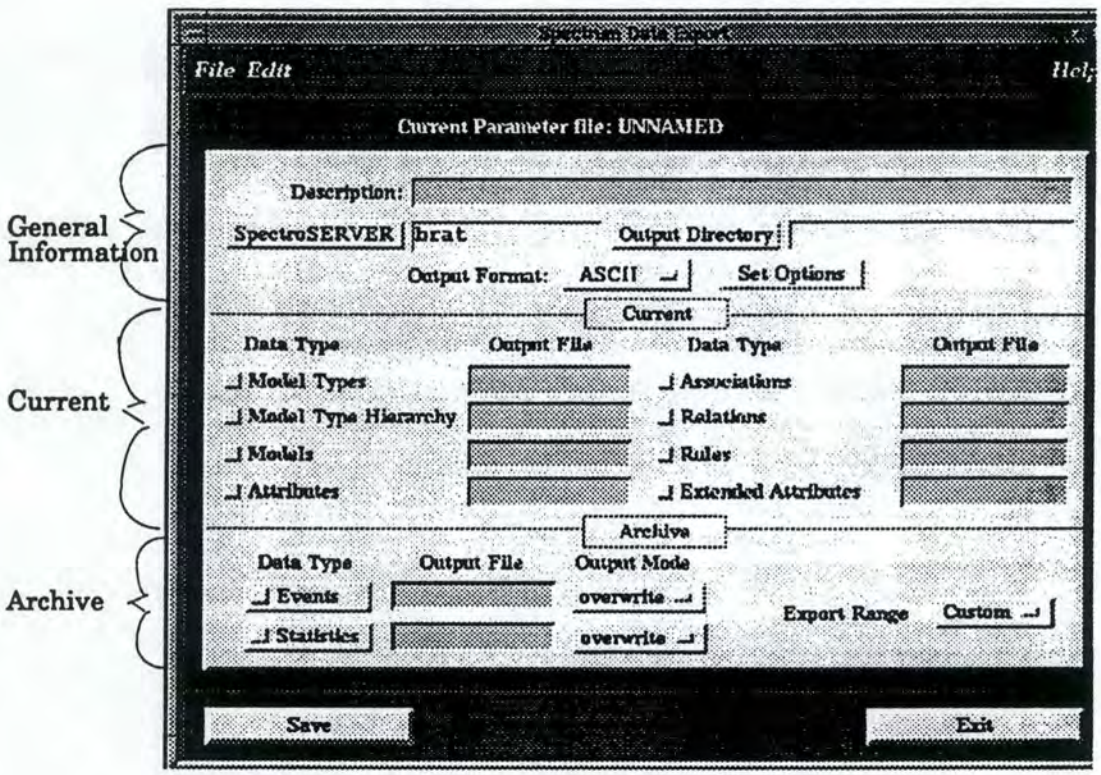


Figure 4.1. Fenêtre de saisie pour la définition du "fichier de sortie" de la fonction DATAEXPORT.

2.1.3. Le format du "fichier de sortie"

Le format du "fichier de sortie" est défini par l'utilisateur. Il doit indiquer quels sont les caractères qui seront utilisés pour séparer, d'une part les champs de données et d'autre part les données elles-mêmes (les "records"). Ces caractères peuvent être des blancs, des virgules, des apostrophes, des tabulations, des caractères indiquant une fin de ligne... ou encore tout autre caractère défini par l'utilisateur.

Dans notre cas spécifique, le format du "fichier de sortie" sera le suivant:

- la virgule pour séparer les différents champs d'une donnée;
- le symbole "New Line" pour séparer les données.

2.1.4. Les critères qui serviront de filtres

Les critères qui seront employés afin de filtrer les informations à exporter peuvent être divisés en trois classes qui sont représentées sous forme de liste:

- **le type de modèle** : la liste contient tous les modèles instanciables qui sont définis dans la base de données du SpectroSERVER. On ne peut sélectionner qu'un seul type de modèle dans la liste (exemples: toutes les interfaces des équipements, tous les routeurs d'une marque particulière, tous les modèles correspondant aux salles existantes...);

- **les modèles** : la liste regroupe tous les modèles qui existent pour le type de modèle donné. L'utilisateur peut en sélectionner plusieurs;

- **les variables de la MIB** : la liste donne toutes les variables de la MIB présentes dans les modèles sélectionnés. Ainsi, l'utilisateur peut sélectionner les variables dont il souhaite obtenir les valeurs lors de chaque "polling". Dans la suite de ce mémoire, nous qualifierons d'"attributs" les variables de la MIB sélectionnées.

2.1.5. Un intervalle de temps

La fonction "DATAEXPORT" extrait des informations de la base de données du SpectroSERVER correspondant à un intervalle de temps précis. L'utilisateur doit par conséquent indiquer l'instant de départ et l'instant de fin (exprimés en mois, jour, année, heures et minutes) qui expriment un intervalle de temps durant lequel il souhaite exporter les informations contenues dans la base de données.

La figure 4.2. illustre la fenêtre du SpectroGRAPH par laquelle l'utilisateur spécifie l'intervalle de temps sur lequel doit porter l'exportation des données.

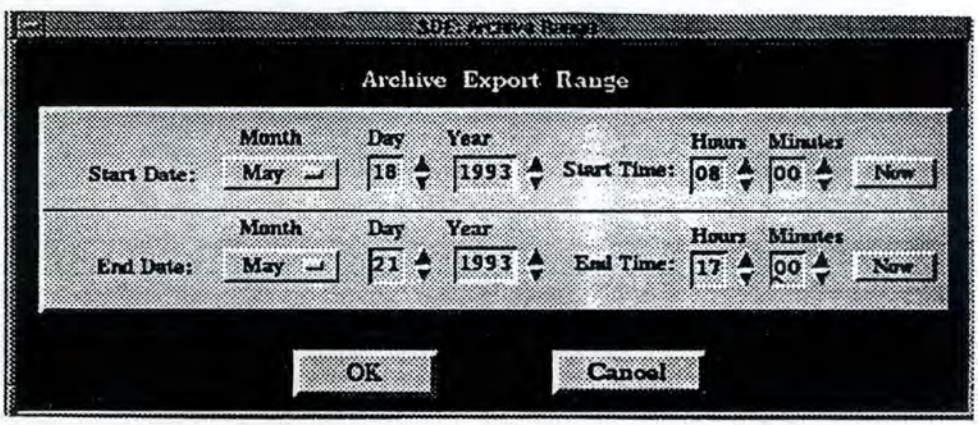


Figure 4.2. Fenêtre du SpectroGRAPH pour spécifier l'intervalle de temps de l'exportation des informations dans le fichier de sortie.

2.1.6. La façon de traiter le "fichier de sortie" précédemment créé

Lors de chaque nouvelle exportation se rapportant toujours à un même "fichier de définition", les informations extraites sont écrites dans le même "fichier de sortie". Mais cette écriture peut s'effectuer de deux manières: soit le "fichier de sortie" précédent est écrasé et les anciennes informations sont remplacées par les nouvelles, soit les nouvelles informations sont ajoutées à la fin du "fichier de sortie" précédent.

2.2. L'exportation des informations

Nous venons de voir que le SpectroGRAPH met à la disposition de l'utilisateur des écrans de saisie pour entrer toutes les données nécessaires à la création du "fichier de définition". Ces écrans permettent de créer, sauver, charger et modifier le "fichier de définition" pour la fonction "DATAEXPORT".
Une fois le "fichier de définition" créé, l'utilisateur peut entreprendre l'exportation des informations qui sont alors écrites dans le "fichier de sortie".

Chaque ligne de notre "fichier de sortie" possède le format suivant:

- la date du "polling" (mm/jj/aa) suivi d'un caractère blanc;
- l'heure du "polling" (hh:mm:ss) suivi d'une virgule et de deux caractères blancs;
- l'identifiant de l'interface qui a fait l'objet du "polling", sous forme hexadécimale, et suivi d'une virgule;
- les valeurs des différents attributs désirés séparées chacune par une virgule;
- le caractère "New Line".

Voici un exemple d'une ligne du "fichier de sortie":

10/11/95 11:37:51, 0x40034a,0000000124,0000004387,0000111653 [NL]

Ce fichier est trié par ordre chronologique. Une interface y apparaîtra, par conséquent, autant de fois que le nombre de "pollings" opérés durant l'intervalle de temps défini dans le "fichier de définition".

Mais cette manière de procéder présente un gros inconvénient: à chaque nouveau "fichier de sortie" correspond un nouveau "fichier de définition". Et ce dernier n'est apparemment pas

générable automatiquement puisqu'il est nécessaire de lui donner un intervalle de temps au moment de sa création. Ceci exige par conséquent une intervention manuelle.
Ainsi, avant de pouvoir visionner un graphique sur les dernières performances du réseau grâce à l'application "*Lego-Plot*", l'opérateur devra tout d'abord ouvrir une session d'un SpectroGRAPH, éditer le "fichier de définition", donner un nouvel intervalle de temps et lancer la fonction "DATAEXPORT" pour obtenir le "fichier de sortie".

Cependant, après quelques investigations, nous avons trouvé une solution à ce problème. Cette solution sera expliquée ultérieurement au point 4.1. de ce chapitre.

3. Le contenu du "fichier d'entrée du "Lego-Plot"

Le "fichier d'entrée du *Lego-Plot*" contient toutes les informations nécessaires pour permettre au "*Lego-Plot*" de réaliser des graphiques sur les performances journalières du réseau. Ce fichier doit comprendre, pour chaque tranche horaire contenue dans une journée, une ligne identifiant la tranche horaire et pour chaque interface, le détail de ses performances.

Ce fichier est construit par le programme "LectureDonnees" qui lui assigne un nom en fonction du jour du mois où les informations contenues dans ce fichier ont été exportées, c'est-à-dire le jour suivant celui auquel les informations correspondent puisque, pour rappel, une exportation réalisée un jour [j] extrait les informations relatives au jour [j-1]. Son nom sera toujours de format "TABLE.LOGnn" où "nn" représente le jour du mois où les informations ont été extraites.

3.1. La syntaxe d'une ligne identifiant une tranche horaire

La ligne qui identifie une nouvelle tranche horaire commencera toujours par la proposition "Information dumped at...". Elle aura la syntaxe suivante:

"Information dumped at 10-Mar-95 14:00:35"

Cette ligne indique la date ainsi que la tranche horaire dans la journée auxquelles correspondent les informations qui suivent dans le fichier (dans cet exemple-ci, il s'agit de la tranche horaire de 14 heures à 15 heures de la journée du 10 mars 1995). Les minutes et secondes qui apparaissent après correspondent au moment où a été réalisé le premier "polling" dans cette tranche horaire. La fin d'une tranche horaire est signalée par un séparateur formé de traits pointillés.

La date à laquelle correspondent les informations exportées est trouvée dans un fichier spécial créé par la fonction "DATAEXPORT", le fichier "exportdata.log". En effet, lors de l'exportation des informations de la base de données du logiciel, la fonction "DATAEXPORT"

écrit dans le fichier "exportdata.log" la manière dont s'est déroulée l'exportation, ainsi que les heures de début et de fin et la date de celle-ci, si toutefois celle-ci a bien eu lieu. A chaque nouvelle exportation, les informations sont ajoutées à la fin de ce fichier. Afin d'éviter qu'il ne s'accroisse indéfiniment, on donnera l'instruction au système d'exploitation d'effacer le fichier "exportdata.log" par exemple tous les lundis matin. Pour cela, il suffit d'écrire l'instruction ainsi que le moment auquel on désire qu'elle soit exécutée dans le fichier "crontab" du système d'exploitation de la machine.

La tranche horaire est, quant à elle, trouvée à partir du "fichier de sortie" créé par la fonction "DATAEXPORT". En effet, ce fichier est trié sur l'ordre chronologique dans lequel les "pollings" ont été réalisés. Ainsi, dans le cas où l'exportation des informations porte sur un intervalle de temps de 24 heures, les "n" premières lignes du fichier (où "n" représente le nombre d'interfaces qui font l'objet d'un "polling" multiplié par le nombre de "pollings" qui ont été effectivement opérés sur chacune d'entre elles durant la tranche horaire) contiendront des informations relatives à des "pollings" effectués entre minuit et une heure du matin. Il suffit donc, pour indiquer la tranche horaire à laquelle se rapporte les informations qui suivent, de prendre le moment du premier "polling" d'une tranche horaire et de lui ajouter "artificiellement" une heure pour rester cohérent avec la signification du mot "dumped" qui signifie "recueillies avant".
En effet, si nous prenons un "polling" effectué dans la tranche des 14 heures à 14:41:54, l'information obtenue grâce à ce "polling" ne peut pas avoir été "recueillie avant" 14 heures.
Donc, dans le cas où le premier "polling" de la tranche horaire entre 14 et 15 heures a été opéré à 14:00:35, l'ensemble des informations concernant cette même tranche horaire sera précédé par une ligne telle que:

"Information dumped at 10-Mar-95 15:00:35"

Pour la tranche horaire allant de 23 heures à minuit, l'ajout d'une unité à l'heure située dans la ligne identifiant la tranche horaire n'entraîne pas une modification de la date c'est-à-dire le passage au jour suivant.

Lorsque l'écriture des informations concernant une tranche horaire est terminée, le programme ajoute une ligne de traits pointillés en guise de séparation des tranches horaires, sauf s'il s'agit de la dernière tranche horaire.

3.2. Les détails des performances de chaque interface

Les informations qui suivent la ligne identifiant la tranche horaire sont relatives aux interfaces qui ont fait l'objet d'un "polling" durant la tranche horaire précédente. Par tranche horaire, chacune de ces interfaces correspondra à une ligne du fichier qui reprendra les informations suivantes:

- le nom du bridge ou du routeur auquel est connectée l'interface (l'interface "*intérieure*");

- le nom du segment auquel appartient l'interface (l'interface "*extérieure*");

Pour chaque attribut désiré:

- sa plus grande valeur sous la forme exponentielle (ex: 4.3 E2) suivie d'un caractère blanc;
- l'heure à laquelle a été enregistrée cette valeur (hh:mm) suivie de deux caractères blancs;

L'ensemble comprenant la valeur d'un attribut et l'heure à laquelle elle a été enregistrée correspond à ce que nous appellerons une colonne du "fichier de sortie".

Et pour indiquer la fin d'une ligne:

- le caractère "New Line".

Exemple : B3Bat31 32UA 2.32 E2 2:34 0.04 E2 2:01 2.45 E3 2:54 [NL]
(1ère col.) (2e col.) (3e col.)

Pour représenter ses différents graphiques, le "*Lego-Plot*" a besoin d'informations diverses. Il s'agit de différentes valeurs des attributs qui seront spécifiés dans le "fichier de définition" de la fonction "DATAEXPORT". Voici ces attributs:

- Le nombre de paquets erronés entrant et sortant par le port d'une interface;
- Le nombre de paquets erronés entrant par le port d'une interface;
- Le nombre de paquets transitant par le port d'une interface;
- Le nombre total d'octets transitant par une interface;
- Le nombre de paquets émis en multi-broadcast par le port d'une interface;
- Le nombre de paquets reçus en multi-broadcast par le port d'une interface.

On devra également se charger d'ajouter deux colonnes supplémentaires dans le "fichier d'entrée du *Lego-Plot*". En effet, le "*Lego-Plot*" a été prévu pour représenter des graphiques en fonction de huit valeurs possibles. Comme actuellement, seulement six des huit valeurs sont utilisées, les colonnes auxquelles n'est assigné aucun des six attributs, à savoir la première et la dernière dans notre "fichier d'entrée du *Lego-Plot*", auront des valeurs et des heures fictives. Ainsi,

la valeur de l'attribut et l'heure écrites dans ces deux colonnes seront respectivement égales à zéro (0.00 E0) et à 00:00.

4. La conception du programme "LectureDonnees"

Le programme "LectureDonnees" constitue la base de notre premier projet au CERN. Il s'occupe de créer le "fichier d'entrée du *Lego-Plot*" à partir des informations extraites de la base de données de SPECTRUM figurant dans le "fichier de sortie" de la fonction "DATAEXPORT". Pour réaliser cela, il analyse les informations contenues dans le "fichier de sortie" pour ensuite les écrire selon un certain format dans le "fichier d'entrée du *Lego-Plot*".

Nous allons maintenant expliquer les différents problèmes rencontrés lors de la conception de ce programme ainsi que les solutions mises en oeuvre.

4.1. La fonction "DATAEXPORT"

Nous venons de voir au point 2.2. que pour activer la fonction qui exporte les informations de la base de données, il nous faut créer un "fichier de définition". Une fois la fonction activée, les informations sont écrites dans un "fichier de sortie". Mais à chaque nouveau "fichier de sortie" correspond un nouveau "fichier de définition". Et à première vue, ce dernier n'est pas générable automatiquement puisqu'il est nécessaire de lui donner un nouvel intervalle de temps au moment de sa création si on ne désire pas toujours obtenir comme résultat un nouveau "fichier de sortie" contenant exactement les mêmes informations.

Afin d'essayer d'améliorer cette méthode lourde demandant une intervention manuelle, il serait indispensable d'arriver à générer automatiquement un nouveau "fichier de sortie".

Après quelques recherches, nous avons trouvé qu'il existait une option dans la fonction "DATAEXPORT" pour générer un "fichier de sortie" tous les jours, sans devoir créer un nouveau "fichier de définition". Ce "fichier de sortie" contient les informations relatives au jour précédent, pour l'ensemble des 24 heures. De plus, la fonction "DATAEXPORT" peut être exécutée par une commande spéciale sans devoir utiliser un SpectroGRAPH. Il suffit donc de sélectionner cette option lors de la création du "fichier de définition" et d'écrire une instruction dans le fichier "crontab" du système d'exploitation pour que ce dernier exécute la fonction "DATAEXPORT" tous les jours à une heure donnée.

Si, par exemple, nous sommes le 24 du mois, cette option aura pour effet l'extraction des informations depuis l'instant 0h 00m 00s jusqu'à l'instant 23h 59m 59s de la journée du 23 de ce même mois.

La fonction "DATAEXPORT" demande pas mal de travail de la part du processeur de la machine sur laquelle elle est exécutée. Il est donc préférable de déclencher l'exportation durant la nuit afin de gêner le moins d'utilisateurs possible.

4.2. Les fichiers des interfaces et des connexions

Le programme "LectureDonnees" travaille à partir de deux fichiers qui décrivent une partie de la situation de la base de données de SPECTRUM. Il s'agit du fichier contenant toutes les interfaces "*extérieures*" accompagnées de leur identifiant et du fichier contenant toutes les connexions de ces interfaces "*extérieures*" avec un équipement tel qu'un bridge ou un routeur, les interfaces "*intérieures*". Dans le cas où ces fichiers s'avèrent statiques, c'est-à-dire qu'ils ne sont pas regénérés lors de chaque nouvelle exécution du programme, toute personne modifiant des informations de la base de données du logiciel relatives à ces fichiers, par exemple en rajoutant une interface d'un nouvel équipement, serait susceptible de devoir regénérer manuellement ces deux fichiers.

Pour être rendu plus flexible envers les modifications qui pourraient survenir dans l'environnement du logiciel SPECTRUM, le programme "LectureDonnees" se chargera lors de son lancement de générer d'abord ces deux fichiers avant de les analyser. Pour lui permettre cela, nous allons utiliser une facilité qu'offre SPECTRUM: le "*Command Line Interface*".

Le "*Command Line Interface*" (CLI)

Le logiciel SPECTRUM met à la disposition du gestionnaire une interface de commandes exécutables (*Command Line Interface*) dans le cas où l'utilisation du SpectroGRAPH n'est pas souhaitable comme par exemple lorsqu'on ne désire pas voir apparaître les résultats d'une commande sur un écran. Cette interface fournit un ensemble de commandes qui autorisent l'utilisateur à effectuer des opérations qui sont normalement réalisées par l'intermédiaire du SpectroGRAPH.

Le CLI est également un serveur local qui est chargé de deux fonctions principales:

- maintenir une connexion Internet constante avec le SpectroSERVER afin d'éviter de devoir opérer une connexion et une déconnexion chaque fois que l'on désire exécuter une commande;

- maintenir, pour chaque utilisateur, l'état de son information. Par exemple, si un utilisateur a défini une interface particulière, il n'est pas obligé de la redéfinir lors de l'exécution de chaque nouvelle commande, elle sera reprise par défaut.

Voici quelques-unes des commandes offertes:

- CONNECT : se connecte au SpectroSERVER. Avant d'exécuter une commande quelconque du CLI, l'utilisateur doit ouvrir une connexion avec le SpectroSERVER;

- DISCONNECT : se déconnecte du SpectroSERVER;

- SHOW : affiche l'information à propos d'un modèle se trouvant dans la base de données de SPECTRUM. Cette commande peut être suivie par:

  - **models** : affiche tous les modèles actuellement définis dans SPECTRUM;

  - **associations** : affiche tous les types d'associations entre les modèles (connecter à, adjacent à, fait partie de, gère...);

  - **parents** : affiche les parents dans l'arborescence d'un modèle donné (exemple: un bridge peut être le parent d'une interface appartenant à un segment);

  - **children** : affiche les enfants dans l'arborescence d'un modèle donné (exemple: un segment connecté à un bridge);

  - **attributes** : affiche tous les attributs d'un type de modèle donné;

  Cette commande peut aussi être accompagnée, soit par un identifiant de modèle, soit par un identifiant d'attribut. Ceci permet de n'avoir comme résultat que le modèle ou l'attribut désiré;

- SEEK: trouve un modèle particulier dans la base de données de SPECTRUM;

- CREATE: crée un nouvel objet dans la base de données de SPECTRUM: un modèle, une association, un événement ou une alarme;

- UPDATE: met à jour les attributs d'un modèle.

La figure 4.3. illustre la manière avec laquelle l'utilisateur peut interroger le SpectroSERVER grâce au CLI. Les commandes émises sont reçues par un serveur local qui interroge le SpectroSERVER via le réseau. Dès que le serveur local reçoit la réponse de la part du SpectroSERVER, il la transmet comme résultat à l'utilisateur.

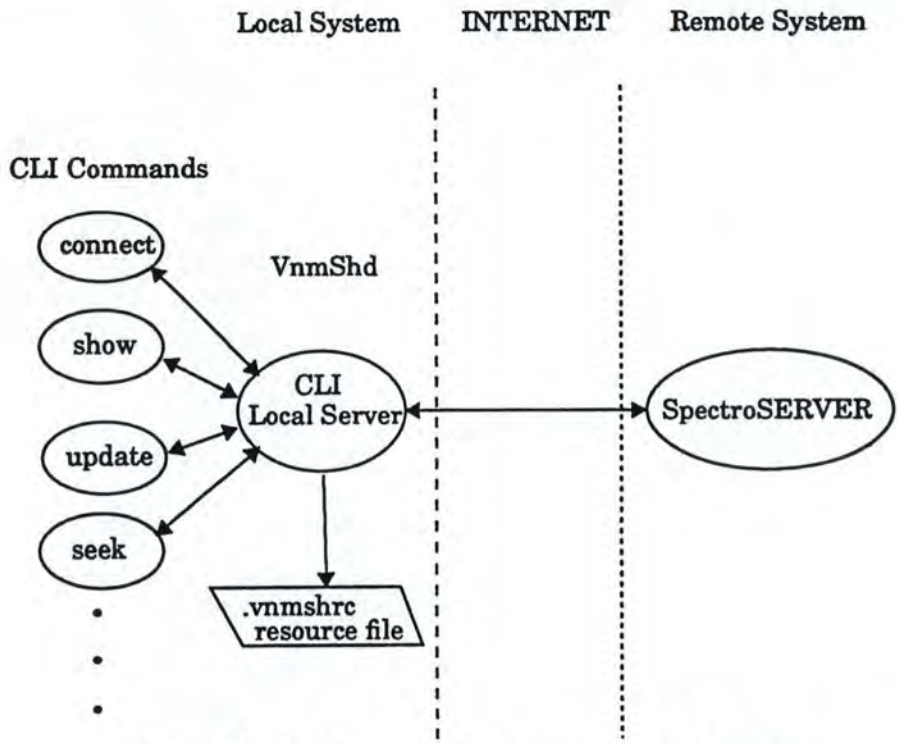


Figure 4.3. Architecture du *Command Line Interface*

L'avantage du "*Command Line Interface*" réside dans le fait que l'on peut faire exécuter des commandes par un programme grâce à des appels "système", sans passer par l'intermédiaire du SpectroGRAPH. Les résultats de ces différentes commandes peuvent alors être redirigés vers ("écrits dans") des fichiers au lieu d'être affichés sur l'écran d'un ordinateur. Ceci se réalise grâce à la commande de "redirection" (>) offerte par le système d'exploitation UNIX.
Nous pouvons par exemple créer un fichier contenant tous les noms des bridges et des routeurs ainsi que le numéro des interfaces "*intérieures*" qu'ils contiennent. En parcourant ce fichier séquentiellement, nous prendrons en compte toutes les interfaces "*intérieures*" des bridges et routeurs définies dans la base de données de SPECTRUM.

4.3. Les attributs des modèles correspondant aux informations nécessaires pour le "*Lego-Plot*"

Nous avons vu que le "*Lego-Plot*" a besoin de certaines valeurs pour réaliser ses différentes représentations graphiques. Ces valeurs correspondent à des attributs (variables de la MIB) qui sont "logués" au sein de chaque agent devant être géré. Ces valeurs d'attributs sont obtenues par le logiciel SPECTRUM grâce au mécanisme de "polling".

A cette fin, il convient tout d'abord de trouver dans l'arborescence de la MIB l'identifiant des attributs qui correspondent aux informations nécessaires au "*Lego-Plot*". Pour cela, nous

allons encore nous servir du "*Command Line Interface*". Un "SHOW attributes" nous donne la liste de tous les attributs se trouvant dans la MIB pour un modèle donné (un bridge, un routeur...). Une fois que l'on a sélectionné les identifiants des attributs, il suffit de les indiquer dans le "fichier de définition" de la fonction "DATAEXPORT".

Les attributs correspondant aux données pour le "*Lego-Plot*" doivent être impérativement des attributs dont la valeur constitue une variation (un delta), c'est-à-dire des attributs dont la valeur est réinitialisée à zéro après chaque "polling". Si tel est le cas, il suffit de prendre la valeur de l'attribut présente dans le "fichier de sortie" et de l'écrire sans la modifier dans le "fichier d'entrée du *Lego-Plot*".

Mais deux de ces attributs ne sont pas réinitialisés à zéro. Il s'agit:

- du nombre de paquets multicast & broadcast émis par seconde;
- du nombre de paquets multicast & broadcast reçus par seconde.

Il sera dès lors nécessaire de garder pour ces deux attributs, leur précédente valeur dans une variable en mémoire centrale. De cette manière, on obtiendra en soustrayant de la nouvelle valeur la précédente, la variation de la valeur pour chacun des deux attributs.

4.4. L'obtention de la date à laquelle correspondent les informations exportées

Pour créer le "fichier d'entrée du *Lego-Plot*", le programme "LectureDonnees" a besoin de la date à laquelle correspondent les informations exportées. Nous avons dit que cette date se trouve dans un fichier créé par la fonction "DATAEXPORT" qui se nomme "exportdata.log".
Mais pour effectuer cette opération, nous avons rencontré un problème.
En effet, la ligne contenant l'information voulue s'avère la quatrième avant la fin du fichier. Or, un fichier ne peut être lu à l'envers et de plus, la taille de notre fichier en question n'est pas fixe. Il faut donc pour obtenir cette ligne, conserver en permanence quatre lignes du fichier durant toute sa lecture. On dispose ainsi d'une sorte de "fenêtre déroulante" d'une hauteur de quatre lignes, située en mémoire centrale.

Au départ, les quatre premières lignes du fichier sont placées dans les quatre lignes de la fenêtre. Lorsqu'on arrive à la cinquième ligne du fichier, celle-ci remplace la première ligne de la fenêtre. La sixième remplacera la deuxième, la septième remplacera la huitième, et ainsi de suite jusqu'à la fin du fichier. Ainsi lorsqu'on arrive à la fin du fichier, on retrouve parmi les quatre lignes situées dans la fenêtre en mémoire centrale celle qui correspond à la quatrième ligne avant la fin du fichier d'après la manière dont s'est terminée la lecture.

Pour illustrer ce mécanisme de la "fenêtre déroulante", considérons un exemple dans lequel la dernière ligne du fichier se retrouve placée dans la troisième ligne de la fenêtre située en mémoire centrale. Ainsi, la deuxième ligne avant le fin du fichier est située, quant à elle, dans la

deuxième ligne de la "fenêtre déroulante", la troisième avant la fin du fichier dans la première ligne de la "fenêtre déroulante" et la ligne souhaitée, à savoir la quatrième avant la fin du fichier, se situe dans la quatrième ligne de la fenêtre puisqu'il n'existe évidemment pas de ligne "zéro".

4.5. La grande taille du "fichier de sortie"

Le "fichier de sortie" de la fonction "DATAEXPORT" a une taille qui varie en fonction de quatre éléments: le nombre d'interfaces faisant l'objet d'un "polling", l'intervalle de temps qui sépare deux "pollings" sur une même interface, le nombre de valeurs d'attributs demandés et le nombre de tranches horaires figurant dans l'intervalle de temps indiqué dans le "fichier de définition". Dans le cas du CERN, le nombre d'interfaces est relativement important et celles-ci sont contrôlées toutes les cinq minutes environ. Pour chacune d'entre elles on demande six valeurs d'attributs et l'intervalle de temps sur lequel portent les informations est de 24 heures. Ce fichier contient dès lors généralement une très grande quantité d'informations.

Un fichier d'une telle taille présente deux désavantages.
Le premier désavantage réside dans le temps de son traitement intégral. En effet, ce fichier peut contenir un nombre très important de lignes. Comme le programme parcourt séquentiellement le fichier autant de fois qu'il y a d'interfaces, le temps d'exécution du programme est inacceptable surtout lorsqu'on sait que la lecture d'une ligne à partir d'un disque demande un temps d'environ 30 milli-secondes. Pour faire face à ce premier problème, nous avons décidé de décharger le "fichier de sortie" dans une table en mémoire centrale.
Le deuxième désavantage provient du fait qu'un tel fichier exige un grand espace disque.

De plus, si nous adoptons la solution qui consiste à décharger le "fichier de sortie" en mémoire centrale pour annuler quasiment les temps de lecture, ce deuxième désavantage sera d'autant plus grand puisque la mémoire centrale d'un ordinateur est une ressource chère et limitée.

Pour essayer de trouver un compromis, nous avons pris la décision de ne décharger le "fichier de sortie" que partiellement en mémoire centrale. Au départ, la première tranche horaire de ce fichier est mise dans la table en mémoire centrale et y reste tant que toutes les informations appartenant à cette tranche horaire n'ont pas été traitées.
Une fois ce traitement terminé, la tranche horaire suivante est chargée en mémoire centrale à la place de la précédente. On reconduit ce mécanisme jusqu'à ce que toutes les tranches horaires aient été traitées.

4.6. La comparaison des "strings"

Durant son exécution, le programme "LectureDonnees" effectue un grand nombre de comparaisons entre les identifiants des diverses interfaces. Ces identifiants représentent des entiers mis sous la forme hexadécimale (exemple: 0x40032a).

Lorsque notre programme lit dans le "fichier de sortie" un identifiant d'une interface écrit sous la forme hexadécimale, il le lit comme un "string" (une chaîne de caractères). Par conséquent, comparer deux identifiants revient à comparer deux "strings". Cet opération prend beaucoup plus de temps qu'une comparaison de deux entiers puisqu'elle demande une comparaison caractère par caractère. Et vu le nombre élevé de comparaisons effectuées tout au long du programme, il était opportun de développer une méthode qui diminuerait le temps de comparaison entre deux identifiants d'interfaces.

La méthode que nous avons développée est la suivante:
nous avons écrit une procédure appelée "atoX" qui s'occupe de transformer un "string" représentant un nombre sous la forme hexadécimale en un entier. Avec cette procédure, tous les identifiants des interfaces sont considérés par le programme "LectureDonnees" comme des entiers, ce qui permet de les comparer en une seule instruction machine. C'est pour cette raison que les identifiants de toutes les interfaces, qui se trouvent sous une forme hexadécimale dans le "fichier de sortie" de la fonction "DATAEXPORT", sont remplacés par des entiers avant d'être insérés dans la table située en mémoire centrale.

5. Architecture du programme

L'architecture du programme "LectureDonnees" se présente comme suit:

D'abord, il crée en mémoire centrale la table "tbl_Gen" des interfaces "*extérieures*" et la table "tbl_Con" des connexions existant entre ces interfaces "*extérieures*" et les équipements du réseaux (bridges et routeurs). Ensuite, il s'occupe de rechercher la date à laquelle a eu lieu l'exportation des informations contenues dans le "fichier de sortie" créé par la fonction "DATAEXPORT". Après, il décharge ce fichier par tranche horaire dans une table "tbl_Stat" située également en mémoire centrale. Et pour chaque tranche horaire, il va rechercher les plus grandes valeurs des six attributs pour chacune des interfaces "*extérieures*". Enfin, une fois les valeurs trouvées, il les écrit dans le "fichier d'entrée du *Lego-Plot*" selon le format exigé par l'application.

5.1. Mise en mémoire centrale du fichier des interfaces "*extérieures*" et du fichier des connexions

Nous avons vu que le fichier contenant les interfaces "*extérieures*" et le fichier contenant les connexions étaient créés par le programme "LectureDonnees" lors de son exécution pour permettre une plus grande souplesse envers les modifications qui pourraient survenir dans l'environnement du logiciel. Ces créations se réalisent grâce à des appels "système".

L'appel "système" pour le fichier des interfaces est le suivant:

system ("/home/suncoms/spectrum/vnmsh/show models | grep Gen_IF_Port > resultat")

où **"/home/suncoms/spectrum/vnmsh/"** constitue le chemin pour accéder aux différentes commandes offertes par le CLI;

"show models | grep Gen_IF_Port" représente la commande en elle-même. Il s'agit d'obtenir une liste de tous les modèles qui correspondent à des interfaces "*extérieures*" ("Gen_IF_Port");

"> resultat" signifie que le résultat de la commande n'est pas affiché à l'écran, mais bien redirigé dans un fichier nommé "resultat".

Lorsque le fichier "resultat" est créé, le programme le décharge en mémoire centrale dans la table "tbl_Gen". Les différents champs de cette table sont:
l'identifiant de l'interface "*extérieure*" sous la forme d'un entier (obtenu grâce à la procédure "atoX"), l'identifiant de l'interface "*extérieure*" sous la forme hexadécimale et le nom du segment auquel appartient l'interface.

En ce qui concerne les connexions entre interfaces et équipements, le programme procède d'une manière identique à celle précédemment décrite pour créer la table "tbl_Con". Les différents champs de cette table sont:
l'identifiant de l'interface "*extérieure*" sous la forme d'un entier (obtenu grâce à la procédure "atoX") et le nom de l'équipement auquel est connectée l'interface "*extérieure*".
Seul l'appel "système" change:

system ("/home/suncoms/spectrum/vnmsh/show children rel=Connects_to mh=<id> > resultat")

où **"show children rel=Connects_to mh=<id>"** représente la commande en elle-même. Il s'agit d'obtenir l'équipement auquel est connectée l'interface "*extérieure*" identifiée par <id>.

La longueur d'une table en mémoire centrale ne pouvant pas être allouée dynamiquement, les longueurs maximales des deux tables "tbl_Con" et "tbl_Gen" sont contenues respectivement dans les constantes "Max_Ent1" et "Max_Ent2". Ces limites supérieures ont été choisies de manière à, normalement, ne pas rencontrer de problèmes. Si la longueur d'une des tables vient à dépasser sa limite supérieure, le programme indique à l'utilisateur que la longueur de la table est trop petite ainsi que la nouvelle valeur à attribuer à la limite. Il suffit à l'utilisateur de changer cette valeur dans le code source du programme "LectureDonnees.c" et de recompiler ce dernier.

5.2. Recherche de la date à laquelle correspondent les informations exportées

Afin de créer le "fichier d'entrée du *Lego-Plot*", le programme a besoin de la date à laquelle correspondent les informations exportées. Pour l'obtenir, il va devoir lire la ligne du fichier "exportdata.log" qui contient la date désirée. La méthode utilisée a déjà été exposée dans l'énumération des problèmes rencontrés lors de la conception du programme et des solutions adoptées.

5.3. Déchargement en mémoire centrale du "fichier de sortie" par tranche horaire

A partir du "fichier de sortie" de la fonction "DATAEXPORT" qui est trié chronologiquement sur les "pollings", le programme lit séquentiellement les lignes s'y trouvant et remplit la table en mémoire centrale "tbl_Stat" qui contiendra l'ensemble des informations pour une tranche horaire. On y retrouve, au niveau de chaque ligne, la date et le moment auxquels les informations ont été enregistrées, l'identifiant de l'interface "*extérieure*" à laquelle appartiennent ces informations et les différentes valeurs des six attributs. Le programme interrompt la lecture du fichier lorsqu'il rencontre une ligne se rapportant à des informations de la tranche horaire suivante. Il reprendra la lecture quand toutes les informations de la table "tbl_Stat" auront été analysées et pourront ainsi être écrasées par de nouvelles informations.

La longueur maximale de la table "tbl_Stat" est contenue dans la constante "Max_Ent3". Si, comme dans le cas des deux autres tables des interfaces "*extérieures*" et des connexions, la longueur de cette table vient à dépasser sa longueur maximale, le programme indique à l'utilisateur que la longueur de la table est trop petite ainsi que la nouvelle valeur à attribuer à la longueur maximale. L'utilisateur n'a plus qu'à changer cette valeur dans le code source du programme "LectureDonnees.c" et à recompiler ce dernier.

5.4. Recherche des valeurs maximales des six attributs des différentes interfaces "*extérieures*" pour une tranche horaire

Au départ, les six variables contenant les valeurs des six attributs sont initialisées à zéro. Pour tout identifiant d'interface "*extérieure*" se trouvant dans la table des interfaces "*extérieures*", le programme lit toutes les lignes de la table contenant l'ensemble des informations pour une même tranche horaire. Dès qu'il tombe sur une ligne qui se rapporte à l'interface "*extérieure*" recherchée, il regarde la valeur des attributs. La recherche de la valeur maximale se réalise de deux manières différentes selon que l'attribut est réinitialisé ou non après un "polling".

5.4.1. Les attributs réinitialisés lors de chaque "polling"

Chaque fois que la valeur d'un attribut est plus grande que toutes les autres précédemment lues, le programme la garde à la place de l'ancienne dans une variable ainsi que l'heure à laquelle elle a été enregistrée. De sorte qu'à la fin de la lecture de la table, on détienne, pour la tranche horaire en question, les valeurs maximales des six attributs pour chacune des interfaces "*extérieures*" ainsi que les heures auxquelles elles ont été enregistrées. Ensuite, on réitère le processus pour tous les identifiants d'interfaces "*extérieures*" contenus dans la table.

5.4.2. Les attributs non réinitialisés lors de chaque "polling"

Les 5ème et 6ème attributs sont traités d'une manière un peu différente car leur valeur ne constitue pas une variation. Il s'agit, pour rappel, du nombre de paquets multicast & broadcast émis par seconde et du nombre de paquets multicast & broadcast reçus par seconde.

Par conséquent, on est obligé de garder dans une variable en mémoire centrale, la valeur précédente pour chacun de ces deux attributs. Ainsi au départ, les valeurs du 5ème et 6ème attributs d'une interface "*extérieure*" se trouvant dans la première ligne de la table "stat_Day" se rapportant à cette même interface "*extérieure*" sont placées respectivement dans les variables "int_AN_Attr5[0]" et "int_AN_Attr6[0]". Les valeurs du 5ème et 6ème attributs de la 2ème ligne de cette table se rapportant toujours à cette interface "*extérieure*" sont placées, quant à elles, respectivement dans les variables "int_AN_Attr5[1]" et "int_AN_Attr6[1]".
De cette façon, le programme peut calculer la variation en soustrayant la valeur de la première variable de celle de la deuxième et la traiter exactement de la même manière que les valeurs des quatre autres attributs. Si nous prenons le cas du 5ème attribut, il suffit, pour obtenir sa variation, de calculer la différence entre "int_AN_Attr5[1]" et "int_AN_Attr5[0]".
Lorsqu'on arrive à la troisième ligne correspondant toujours aux valeurs de cette interface "*extérieure*", les variables "int_AN_Attr5[0]" et "int_AN_Attr6[0]" sont remplacées par les variables "int_AN_Attr5[1]" et "int_AN_Attr6[1]" qui se voient à leur tour assigner de nouvelles valeurs. On réitère ainsi le processus jusqu'à ce qu'on arrive au bout de la tranche horaire.

Toutefois, avec cette méthode, il peut exister une erreur dans la recherche de la plus grande variation d'un de ces deux attributs pour une interface "*extérieure*". Ceci survient dans le cas où la plus grande variation se produit entre le dernier "polling" d'une interface "*extérieure*" pour une tranche horaire, et le premier "polling" de cette même interface "*extérieure*" pour la tranche horaire suivante. Mais ce léger problème n'est pas assez conséquent pour y remédier, en tout cas dans un avenir proche.

5.5. Recherche de l'équipement auquel est connectée une interface

Lorsqu'on dispose pour une tranche horaire de toutes les valeurs maximales pour une interface "*extérieure*", il s'agit d'obtenir le nom de l'équipement auquel elle est connectée. Pour ce faire, le programme compare l'identifiant de l'interface "*extérieure*" avec ceux contenus dans la table des connexions. Une fois qu'il y a correspondance, il suffit d'aller lire le champ suivant de la table des connexions pour obtenir le nom de l'équipement.

Dans le cas où l'identifiant de l'interface "*extérieure*" n'existe pas dans la table des connexions, on écrit "unnamed" dans la variable devant contenir le nom de l'équipement.

5.6. Ecriture d'une ligne dans le "fichier d'entrée du *Lego-Plot*"

Maintenant que l'on se trouve en possession de tous les éléments composant une ligne du "fichier d'entrée du *Lego-Plot*", il ne reste plus au programme qu'à les écrire selon le format défini au point 3. pour qu'ils soient lisibles par l'application graphique.

6. La phase de test du programme "LectureDonnees"

Le programme "LectureDonnees" a été testé à partir d'un certain "fichier de sortie" de la fonction "DATAEXPORT". Ce fichier, qui fut créé une fois pour toute lors de la réalisation des premiers tests, contenait une quantité de données réduite pour diminuer au maximum le temps d'exécution du programme durant les nombreux tests, mais assez conséquente pour essayer de prendre en considération tous les cas possibles qui pouvaient se présenter.
Après chaque exécution du programme "LectureDonnees", il nous fallait vérifier si le "fichier d'entrée du *Lego-Plot*" fourni comme résultat disposait du bon format pour être lisible par l'application "*Lego-Plot*". Dans la négative, nous devions aller modifier le format du fichier dans le code source du programme, le recompiler et relancer son exécution.

7. Conclusion

Le programme "LectureDonnees" sera exécuté tous les matins et fournira le "fichier d'entrée du *Lego-Plot*" pour que les opérateurs et l'administrateur du réseau puissent lancer à tout moment de la journée le "*Lego-Plot*" et ainsi obtenir le graphique représentant les performances des différents composants du réseau du jour précédent. Cependant, il convient de faire attention à ce que la fonction "DATAEXPORT" soit exécutée avant le lancement du programme "LectureDonnees" pour que les informations contenues dans le "fichier d'entrée du *Lego-Plot*" soient actualisées.

Nous nous permettons d'informer le lecteur que, s'il désire consulter le code source du programme "LectureDonnees", il peut le trouver à l'annexe 3 située à la fin de ce mémoire.

Chapitre 5:
Le système d'alarme

Ce cinquième chapitre sera consacré à la présentation de notre deuxième projet. Nous définirons d'abord le problème posé et nous dirons quelques mots sur la communication inter-processus dont nous nous servirons pour réaliser ce projet. Ensuite, nous expliquerons l'utilitaire "AlarmMonitor" proposé par le logiciel SPECTRUM et sa nécessité pour la réalisation du projet. Pour terminer, nous donnerons une explication sur la manière dont les différents programmes de ce projet ont été conçus.

1. Définition du problème

Nous avons vu que dans un réseau de très grande ampleur, il était indispensable de disposer d'une application capable de détecter tout problème pouvant survenir au niveau d'un composant (bridge, routeur, interface...) et d'en avertir au plus vite les opérateurs pour y remédier. Le logiciel SPECTRUM dispose d'une application qui se charge de l'affichage de toutes les alarmes qu'il a détectées. Cet affichage s'effectue dans la vue "*Alarme*" (voir figure 2.8.). Mais toutefois, cette vue "*Alarme*" ne convient pas à l'administrateur et aux opérateurs du réseau du CERN qui préfèrent garder opérationnelle leur propre application "*AlarmMirror*".

Par conséquent, pour notre deuxième projet, on nous a d'abord demandé de trouver un moyen d'accéder à toutes les informations sur les alarmes actuelles (et futures) qui se trouve(ro)nt dans la base de données de SPECTRUM et ensuite, une fois ces informations accessibles,

d'envoyer les informations concernant chaque alarme à l'application "*AlarmMirror*" via un mécanisme de communication inter-processus.

2. La communication inter-processus

Lorsqu'on parle d'un mécanisme de communication inter-processus, il s'agit en réalité de "sockets" grâce auxquels deux processus, ne tournant pas nécessairement sur une même machine, peuvent s'échanger des informations.

2.1. La création d'un "socket"

Lorsqu'on crée un "socket", on crée en fait un nouveau descripteur de fichier pointant vers un fichier qui servira de "lieu d'échange d'informations".
D'une manière plus pratique, lorsque deux processus désirent communiquer entre eux, ils doivent chacun créer le "socket". Lorsqu'un nom a été choisi pour le "socket", il suffit que l'un des deux processus le lui attribue avec un appel au système ("bind"). Le nom apparaîtra alors dans son répertoire de travail.
Ces "sockets" se trouvent dans le domaine Internet. L'adresse Internet spécifie une adresse d'un hôte sur 32 bits ainsi qu'un numéro de port. Ce port est géré par des routines du système qui implémentent un protocole particulier.

2.2. La communication entre "sockets"

Si un processus désire communiquer de l'information à un autre processus, il l'écrit dans le fichier créé par l'ouverture du "socket". Une fois l'information écrite, le processus qui attend l'information peut alors commencer à la lire.
Le "socket" est un moyen de communication à sens unique, c'est-à-dire qu'il y a un processus qui écrit l'information et un processus qui lit l'information. Si on désire une communication bilatérale, on doit utiliser un autre mécanisme de communication: le "socketpair".

Quand un message doit être communiqué entre deux machines, il est envoyé à la routine qui gère le protocole sur la machine destinataire. Cette routine interprète l'adresse pour déterminer à quel "socket" délivrer le message.
Plusieurs protocoles différents peuvent être présents sur une même machine. Par conséquent, les différents protocoles sont autorisés à employer le même numéro de port. C'est pour cette raison que l'adresse Internet est un triplet comprenant, en plus du numéro de port et de l'adresse de la machine, un protocole qui est choisi lors de la création du "socket".
En réalité, le numéro de port peut être vu comme le numéro d'une boîte aux lettres, dans laquelle le protocole place les messages.

2.3. Les types de communication entre deux sockets

La communication entre deux "sockets" peut être de deux types: de type STREAM ou de type DATAGRAM.

2.3.1. Le type STREAM

Dans le cas d'une communication inter-processus de type STREAM, la communication est réalisée par l'intermédiaire d'une connexion. Cela implique l'ouverture de la connexion avant l'envoi de tout message et la fermeture de la connexion dès que la communication est terminée. Elle est fiable et sans erreur (dans le cas où une erreur apparaîtrait dans le message, ce dernier serait ré-expédié).

2.3.2. Le type DATAGRAM

Par contre, dans le cas d'une communication inter-processus de type DATAGRAM, la communication ne demande pas une connexion préalable. Chaque message est adressé individuellement. Si l'adresse est correcte, il sera très probablement délivré au destinataire, mais ce n'est toutefois pas garanti.

2.3.3. Différence de performance entre type STREAM et type DATAGRAM

La différence de performance entre ces deux types de communication est généralement moins importante que la différence de sémantique. Le gain de performance que l'on pourrait trouver dans l'utilisation des "datagrams" devrait être amoindri par la complexité croissante du programme qui les utilise. En effet, celui-ci doit maintenant veiller lui-même aux pertes de messages. Dans le cas où les messages perdus seraient simplement ignorés, la quantité de trafic devrait être alors prise en considération puisque un même message serait susceptible d'être renvoyé plusieurs fois.

3. L'utilitaire "AlarmMonitor" du logiciel SPECTRUM

Le logiciel SPECTRUM met à la disposition des utilisateurs un utilitaire qui permet de récupérer dans la base de données de SPECTRUM toute information sur une alarme en provenance d'un composant du réseau. Il s'agit de l'utilitaire "AlarmMonitor".

Pour pouvoir lancer cet utilitaire, l'utilisateur doit lui fournir d'une part un fichier "*alarmrc.dat*" dans lequel il a placé certains paramètres et d'autre part, trois programmes "*SetScript*", "*ClearScript*" et "*UpdateScript*" qui sont en fait des scripts qui seront exécutés en fonction du changement d'état d'une alarme.

Voici leur définition:

- alarmrc.dat: fichier contenant plusieurs paramètres nécessaires lors du lancement de l'utilitaire. Ce fichier est localisé dans le répertoire spectrum/SG-Tools.

  Les différents paramètres qu'il contient sont:

  - le nom de la machine hôte;

  - le numéro de socket sur lequel le SpectroSERVER envoie les informations;

  - les noms des trois scripts.

  Par défaut, le fichier "alarmrc.dat" contient ces informations-ci:

  ```
  VNM NAME        = Suncoms
  SOCKET NUMBER   = 0xBEEF
  SET SCRIPT      = SetScript
  CLEAR SCRIPT    = ClearScript
  UPDATE SCRIPT   = UpdateScript
  ```

- *SetScript* : script appelé par l'utilitaire lorsqu'une nouvelle alarme survient;

- *ClearScript* : script appelé par l'utilitaire lorsqu'on a remédié à une alarme;

- *UpdateScript* : script appelé par l'utilitaire lorsqu'une alarme change de condition, c'est-à-dire d'état.

Lorsque l'utilitaire "AlarmMonitor" fait appel à ces trois scripts, il leur passe plusieurs paramètres sous la forme de string:

- **Date** : date à laquelle l'alarme a été détectée;
- **Time** : heure à laquelle l'alarme a été détectée;
- **Mtype** : type du modèle dont provient l'alarme;
- **ModelName** : nom du modèle dont provient l'alarme;
- **AlarmId** : numéro identifiant l'alarme. Cet identifiant est utile pour pouvoir prendre en considération un changement d'état d'une alarme existante;
- **Condition** : couleur représentant l'état d'une alarme. Pour rappel, la signification des couleurs a été donnée au point 3.5.3. du deuxième chapitre;
- **CauseCode** : code sous une forme hexadécimale qui correspond à la cause de l'alarme. Pour connaître ce que signifie ce code, c'est-à-dire à quel problème il correspond d'après l'analyse qu'en a fait SPECTRUM, il faut aller regarder dans un fichier particulier (VPAPI/include/CsAlarmCause.h);
- **RepairPerson** : nom de la personne qui est chargée de remédier à l'alarme (information entrée dans la base de données de SPECTRUM par l'administrateur ou toute autre personne compétente);
- **AlarmStatus** : statut de l'alarme.

C'est par l'intermédiaire du SpectroSERVER que l'utilitaire "AlarmMonitor" retrouve toutes les informations sur les alarmes qui ont été détectées par le logiciel SPECTRUM et informe l'utilisateur de l'occurrence d'une nouvelle alarme.
Quand l'utilitaire "AlarmMonitor" est lancé, il recherche d'abord toutes les informations de la journée sur les alarmes qui se sont passées entre minuit (0h 00m 00s) et l'heure à laquelle il a été lancé. Pour chacune d'entre elles, il lance le script approprié en fonction de l'information qu'il détient à propos de l'alarme.
Ensuite, lorsque toutes ces alarmes ont été traitées, l'utilitaire "AlarmMonitor" attend l'occurrence d'une nouvelle alarme ou un changement d'état d'une alarme existante qui lui sera indiqué par le SpectroSERVER.

Remarque :

Les trois scripts présents dans le fichier "alarmrc.dat" s'occupent seulement d'afficher sur l'écran de l'utilisateur, qui lance l'utilitaire "AlarmMonitor", toutes les informations sur les alarmes qui leur sont passées comme paramètres par le SpectroSERVER.

4. La solution implémentée

En comprenant les moyens qui nous étaient offerts pour obtenir les informations sur les alarmes détenues dans la base de données du logiciel SPECTRUM, grâce à l'utilitaire "AlarmMonitor", nous avons décidé de concevoir trois petits programmes qui seront exécutés séparément. Au moment de leur lancement, chacun de ces programmes recevra en entrée des informations sur la condition d'alarme d'un composant du réseau détectées par le logiciel. Il devra d'abord filtrer et traiter ces informations pour ensuite les envoyer à l'application "*AlarmMirror*" par le mécanisme de communication entre processus, "le socket".

Pour ce faire, il convient tout d'abord de modifier les trois scripts par défaut contenus dans le fichier "alarmrc.dat". Chaque nouveau script fera maintenant appel à l'un de ces trois programmes avec comme paramètres toutes les informations nécessaires sur une alarme.

Vu que les codes de ces trois scripts ne nous sont pas accessibles, nous avons décidé de créer de nouveaux scripts et de modifier le fichier "alarmrc.dat" en conséquence.

Dès lors, voici le choix que nous avons effectué:

- le script "*SetScript*" devient "*SetScript1*". Lorsqu'il est exécuté, il fait appel au programme "Set_Alarm";

- le script "*UpdateScript*" devient "*UpdateScript1*". Lorsqu'il est exécuté, il fait appel au programme "Update_Alarm";

- le script "*ClearScript*" devient "*ClearScript1*". Lorsqu'il est exécuté, il fait appel au programme "Cancel_Alarm".

Les appels réalisés par les scripts aux différents programmes se font avec, comme paramètres, les informations nécessaires sur l'alarme. Ces informations sont:

- le type du modèle qui occasionne une alarme;
- le nom du modèle qui occasionne une alarme;
- l'identifiant de l'alarme;

- le code de la cause de l'alarme;
- la couleur de l'alarme qui représente son état.

5. Explication des programmes

Les programmes "Set_Alarm" et "Update_Alarm" s'occupent tout d'abord de remplir un paquet (une "trame") qu'ils transmettront plus tard via le réseau à l'application "*AlarmMirror*" avec toutes les informations nécessaires sur l'alarme qui a entraîné leur exécution.

Ensuite, en fonction de la couleur de l'alarme c'est-à-dire de son état, un identifiant particulier sera ajouté au début du paquet pour informer l'application "*AlarmMirror*" sur la gravité de l'alarme. Une fois cette opération achevée, le programme crée un "socket" et envoie le paquet à l'application "*AlarmMirror*" via un numéro de port réservé à cet effet. Il s'agit d'un "socket" dont l'adresse de l'hôte se trouve dans le domaine Internet. Le type de communication est le "datagram" avec le protocole UDP (User Datagram Protocol).
Lorsque l'application "*AlarmMirror*" reçoit un paquet en provenance d'un de ces deux programmes, celle-ci regarde l'identifiant de l'alarme contenu dans le paquet. Si l'application ne connaît pas cet identifiant, elle affiche l'information de l'alarme à l'écran.
Par contre, si l'identifiant de l'alarme ne lui est pas inconnu, l'application modifie en conséquence l'information qu'elle détient déjà sur l'alarme.

Le programme "Cancel_Alarm" s'occupe, quant à lui, des alarmes auxquelles on a remédié. Chaque fois qu'une alarme n'a plus de raison d'être, il envoie à l'application "*AlarmMirror*" un paquet avec un identifiant particulier pour l'informer que le message préalablement affiché à propos de cette alarme doit être effacé.

Maintenant que nous avons donné un petit aperçu sur ce que réalisent les trois programmes, nous allons passer à leur explication détaillée.

5.1. Le programme "Set_Alarm"

PARAMETRES : (type,modele,alarm_id,code_cause,coul)

Le programme "Set_Alarm" est constitué d'un corps principal et d'une procédure "atoX" qui transforme un nombre sous la forme hexadécimale en un nombre sous la forme décimale, ceci afin d'optimiser le temps de comparaison entre deux codes de cause d'alarme. Ainsi, lorsqu'on aura un string représentant un entier sous la forme décimale, il ne nous restera plus qu'à le transformer en entier grâce à l'instruction du langage C "atoi".

Ce programme est lancé lorsqu'une nouvelle alarme survient. Il analyse d'abord de quel type est le composant du réseau dont provient l'alarme ainsi que la couleur (l'état) de l'alarme.

C'est en fonction de cette analyse que le programme détermine l'identifiant particulier contenu dans le paquet qu'il enverra à l'application "*AlarmMirror*". Cet identifiant peut prendre deux valeurs: "AL" signifiant "alarme" et "WA" signifiant "warning", c'est-à-dire "avertissement".

Voici les différentes hypothèses devant lesquelles on peut se trouver lorsque le programme reçoit un message d'alarme. Soit l'alarme provient:

- d'un modèle de type "Fanout" (segment)
  .avec une alarme de couleur rouge => identifiant = "AL".
  .avec une alarme de couleur orange => identifiant = "WA".

- d'un modèle de type "WA_Segment" ou "Coax_Segment" dont la cause de l'alarme est différente de "contact perdu" ("Contact Lost": le code de la cause sous une forme décimale = 65545)
  .avec une alarme de couleur rouge => identifiant = "AL".
  .avec une alarme de couleur orange => identifiant = "WA".

- de tout autre type de modèle
  .avec une alarme de couleur rouge ou orange => identifiant = "AL".
  .avec une alarme de couleur jaune ou grise => identifiant = "WA".

Les alarmes de couleur "bleue" sont ignorées.

Par souci d'alléger l'écran d'affichage du système d'alarme, on nous a demandé de ne pas traiter les alarmes concernant toute une série de types de modèle. C'est pour cette raison qu'un premier tri de toutes les alarmes est réalisé en début de programme.
Les types de modèle à ne pas prendre en compte ne nous semblant pas une information pertinente, nous ne pensons pas qu'il soit nécessaire de les énumérer ici. Toutefois pour ceux qui seraient tout de même intéressés par cette information, nous les renvoyons au code source du programme situé en annexe 4 de ce mémoire.

Une fois que le premier tri des alarmes a été effectué, il convient de vérifier que le code de l'alarme n'est pas égal à celui de "Contact Lost" (65545), pour autant que l'alarme ne provienne pas d'un modèle de type "Fanout". A cette fin, on prend simplement le code de la cause de l'alarme sous forme hexadécimale et on le transforme en un code sous forme décimale grâce à la procédure "atoX". Il ne nous reste plus qu'à comparer les deux entiers obtenus grâce à l'instruction "atoi" et qui sont le code et le nombre 65545.

Ensuite, le programme continue à remplir le paquet ("data") avec toutes les informations contenues dans les paramètres (l'identifiant de l'alarme, le type et le nom du modèle) et le nom du système qui a détecté l'alarme, à savoir SPECTRUM.

Le programme "Set_Alarm" a besoin d'un fichier "Alarm_Code". Ce fichier reprend toutes les causes d'alarme connues par le logiciel SPECTRUM ainsi que leur code respectif. Il s'agit

simplement d'une copie du fichier "VPAPI/include/CsAlarmCause.h" dans laquelle les codes des causes des alarmes ne sont plus sous une forme hexadécimale, mais bien sous une forme décimale (ceci afin d'optimiser la comparaison entre les codes contenus dans le fichier et le code de l'alarme).
Par conséquent, en parcourant le fichier "Alarm_Code", on trouvera normalement l'explication du code de l'alarme. Dès qu'elle a été trouvée, on la place également dans le paquet.
Si le code de l'alarme donné comme paramètre ne correspond à aucun code du fichier "Alarm_Code", on met dans le paquet "Unknown cause" (cause inconnue) comme explication.

Lorsque le paquet est rempli avec toute l'information nécessaire, un socket est ouvert avec une adresse dans le domaine Internet et le protocole UDP. Le paquet est alors envoyé sur le numéro de port 3210 vers la machine sur laquelle tourne l'application "*AlarmMirror*".

5.2. Le programme "Update_Alarm"

PARAMETRES : (type,modele,alarm_id,code_cause,coul)

Le programme "Update_Alarm" est lancé pour toute modification d'état d'une alarme existante. Il utilise également une procédure "atoX", identique à celle qui se trouve dans le programme "Set_Alarm".

Comme les messages se rapportant à des modifications de l'état d'une alarme qui sont envoyés à l'application "*AlarmMirror*" doivent se rapporter à des alarmes existantes et prises en compte par le système d'alarme, le programme "Update_Alarm" effectue les mêmes tests et les mêmes opérations que le programme "Set_Alarm".

Si nous avons décidé de créer deux programmes séparés, "Set_Alarm" et "Update_Alarm", mais identiques (pour le moment en tout cas), c'est dans un souci de clarté et de flexibilité. En effet, il est tout d'abord plus clair pour la compréhension du processus de disposer d'un programme propre à chaque script. Ensuite, rien ne dit que dans un futur plus ou moins éloigné, ces deux programmes seront encore exactement semblables. Il suffira alors de prendre l'un des deux codes sources et de le modifier indépendamment de l'autre.

5.3. Le programme "Cancel_Alarm"

PARAMETRES: (type,modele,alarm_id,code_cause,coul)

Le programme "Cancel_Alarm" est lancé pour toute annulation d'une alarme existante. Tout comme dans le cas des messages de modification de l'état d'une alarme, les messages d'annulation d'alarme envoyés à l'application "*AlarmMirror*" doivent également se rapporter à des alarmes existantes et prises en compte par le système d'alarme. Par conséquent, le programme "Cancel_Alarm" effectue les mêmes tests que le programme "Set_Alarm" pour trier tous les messages d'annulation.

Ensuite, ce programme remplit le paquet ("data") avec un identifiant de paquet égal à "CA" ("cancel", signifiant annulation), le nom du système qui a envoyé le message à savoir SPECTRUM, l'identifiant de l'alarme, le type et le nom du modèle qui a occasionné l'alarme. Ce paquet est alors envoyé à l'application "*AlarmMirror*" via le socket venant juste d'être créé.

Remarque :

Pour que ces trois programmes soient utilisés lorsqu'un utilisateur lance l'exécution de l'utilitaire "AlarmMonitor" proposé par le logiciel SPECTRUM, il convient de modifier les chemins d'accès des scripts se trouvant dans le fichier "alarmrc.dat" qui se situe dans le répertoire [~spectrum/SG-Tools/] comme suit:

VNM NAME = Suncoms

SOCKET NUMBER = 0xBEEF

SET SCRIPT = /home/suncoms/lplattea/Projet_Spectrum/ALARM_MIRROR/SetScript1

CLEAR SCRIPT =
/home/suncoms/lplattea/Projet_Spectrum/ALARM_MIRROR/ClearScript1

UPDATE SCRIPT =
/home/suncoms/lplattea/Projet_Spectrum/ALARM_MIRROR/UpdateScript1

6. La phase de test des différents programmes

Pour tester les différents scripts, on nous a mis à disposition un bridge à deux interfaces qui n'avait plus aucune utilité dans le réseau. Nous pouvions ainsi "jouer" avec ce bridge en simulant une perte de contact avec une de ses interfaces ou en coupant tout simplement son alimentation. De cette façon, après avoir lancé l'utilitaire "AlarmMonitor" de SPECTRUM, nous arrivions à vérifier si les informations sur l'alarme que nous venions de déclencher étaient bien transmises à "*AlarmMirror*" et correctes.

Mais chacun des tests prit pas mal de temps pour deux raisons. La première raison se situe dans le fait que lorsqu'on lance l'utilitaire "AlarmMonitor", toutes les alarmes détectées par SPECTRUM depuis minuit (0h 00m 00s) se trouvant dans sa base de données sont d'abord recherchées. Dès lors, vu que nous lancions "AlarmMonitor" au plus tôt à 8 heures du matin, il nous était nécessaire d'attendre un certain temps avant que l'utilitaire ait parcouru toutes les alarmes contenues dans la base de données et puisse enfin transmettre l'information sur l'alarme venant d'être déclenchée.
Le deuxième raison provient de l'intervalle de temps séparant deux "pollings" successifs sur un même équipement. Nous avons dit que cet intervalle était d'environ 300 secondes au CERN, de sorte qu'il nous fallait attendre en moyenne 150 secondes pour que l'alarme que nous venions de déclencher soit détectée par le SpectroSERVER.

7. Conclusion

Bien que le logiciel SPECTRUM dispose de sa propre vue "*Alarme*", les opérateurs du CERN ont préféré garder opérationnelle l'application "*AlarmMirror*". C'est pour cette raison que les programmes "Set_Alarm", "Update_Alarm" et "Cancel_Alarm" ont été conçus et peuvent être considérés comme des interfaces entre le logiciel et l'application. Leur but est simplement de transférer de l'information récoltée par SPECTRUM sur les alarmes du réseau vers l'application "*AlarmMirror*". Nous pouvons cependant ajouter que la présence de l'utilitaire "AlarmMonitor" nous a facilité la réalisation de ce projet.

Nous nous permettons d'informer le lecteur que, s'il désire consulter les codes sources des programmes "Set_Alarm.c", "Update_Alarm.c" et "Cancel_Alarm.c", il peut les trouver à l'annexe 4 située à la fin de ce mémoire.

Chapitre 6:
La configuration des vues "*Localisation*"

Dans ce sixième chapitre, nous allons présenter notre troisième et dernier projet "Location_View" se rapportant à la configuration des vues "*Information*" et des vues "*Localisation*" de SPECTRUM. Pour commencer, nous rappellerons ce que sont les vues "*Localisation*", les vues "*Information*" et les vues "*Topologie*" et nous poserons les grandes bases du projet. Ensuite, nous en expliquerons ses deux parties ainsi que les méthodes qui nous étaient offertes pour sa réalisation, accompagnées de leurs inconvénients et avantages respectifs. Pour finir, nous développerons comment les six programmes constituant le projet ont été conçus.

1. Définition de problème

Nous avons vu dans le deuxième chapitre que le logiciel SPECTRUM offre des vues "*Localisation*", des vues "*Information*" et des vues "*Topologie*".

Les vues "*Localisation*" constituent des vues hiérarchiques et représentent différents "emplacements physiques" (des lieux). Ainsi, au niveau du CERN, on peut disposer d'une vue d'ensemble sur l'étendue du site divisé en différentes zones. Ensuite, en allant cliquer sur une zone, on peut apercevoir l'ensemble des bâtiments situés dans cette zone. Et ainsi de suite, on peut obtenir une vue représentant un bâtiment particulier avec ses salles et une salle particulière avec ses "compartiments". Les vues "*Localisation*" correspondant à des bâtiments ou à des salles peuvent comprendre des icônes symbolisant les équipements du réseau se trouvant physiquement dans ces bâtiments ou ces salles.

Les vues "*Topologie*" sont également des vues hiérarchiques et représentent la topologie du réseau basée sur les connexions qui existent entre les composants. Elles contiennent des icônes symbolisant des équipements (bridges, routeurs, stations de travail...), des sous-réseaux, des segments et les connexions existant entre eux.

Les vues "*Information*", quant à elles, constituent des fenêtres dans lesquelles apparaissent diverses informations sur un composant particulier du réseau (un bridge, un routeur, une interface...). Voici des exemples d'informations que l'on peut trouver sur un composant: son nom, son adresse physique, sa localisation, ses "logged attributes", le temps qui s'est écoulé depuis sa dernière réinitialisation, l'état (la couleur) dans lequel il se trouve, etc...

Avant de définir notre troisième et dernier projet, il est bon de savoir que lorsqu'un opérateur crée le modèle d'un nouvel équipement appartenant au réseau, soit par l'intermédiaire du SpectroGRAPH, soit par le "*Command Line Interface*", l'icône modélisant l'équipement est automatiquement placée dans une des vues "*Topologie*" de SPECTRUM. La vue "*Topologie*" dans laquelle est copié l'équipement dépend de sa position dans le réseau. Par contre, même lorsqu'on fournit de l'information sur la localisation physique de l'équipement (que l'opérateur aura dû préalablement aller chercher dans une base de données particulière), l'icône le symbolisant n'est pas automatiquement copiée dans la vue "*Localisation*" adéquate. C'est à l'opérateur de faire un "copier-coller" de l'icône vers la vue "*Localisation*". Cet inconvénient prend toute son importance lorsqu'on sait que le logiciel peut contenir des milliers de modèles.

Pour en venir à notre projet, il se concentre sur les vues "*Information*" et les vues "*Localisation*" représentant des bâtiments et des salles situés sur le site du CERN. On nous demande d'abord d'aller rechercher dans une base de données ORACLE l'emplacement physique de tout équipement apparaissant dans la base de données de SPECTRUM. Une fois cette localisation trouvée, on doit vérifier si sa syntaxe est conforme à la syntaxe décrite plus loin au point 2.1.1. de ce même chapitre. Dans le cas où la réponse est positive, nous devons initialiser l'attribut "*Location*" de la vue "*Information*" de l'équipement en question avec la valeur de cette localisation. Cet attribut "*Location*" informe, comme son nom l'indique, sur l'endroit où se situe l'équipement en termes de bâtiment et de salle.

Ensuite, lorsque cette première partie du projet est réalisée, nous devons configurer les vues "*Localisation*" correspondant à ces emplacements physiques en les "remplissant" avec les icônes symbolisant les équipements du réseau qui s'y trouvent réellement.

2. Explication des étapes du projet

Nous allons maintenant détailler, en deux parties, la manière avec laquelle nous avons résolu l'entièreté du projet. Nous énoncerons également les problèmes auxquels nous avons été confrontés, les solutions qui nous étaient proposées avec leurs avantages et leurs inconvénients et les choix pour lesquels nous avons opté.

2.1. Première partie

Comme on a pu le lire au début de ce chapitre, la première partie de notre projet comporte deux aspects: la recherche de l'emplacement physique d'un équipement du réseau dans la base de données ORACLE et l'initialisation de l'attribut "*Location*" de sa vue "*Information*".

2.1.1. La recherche de l'emplacement physique d'un équipement

Pour connaître la localisation des équipements figurant dans la base de données de SPECTRUM, il faut interroger la base de données ORACLE. Celle-ci donne, pour tout équipement appartenant au réseau, sa localisation en termes de bâtiment et également, dans beaucoup de cas, de salle. Cette interrogation se réalise par l'intermédiaire d'une requête dans le langage SQL.

Une fois trouvée la localisation d'un équipement dans la base de données ORACLE, on vérifie tout d'abord sa syntaxe. Soit elle est conforme à la syntaxe décrite au paragraphe suivant et dès lors on peut poursuivre le traitement, soit elle est non conforme et nous devons en avertir le gestionnaire de la base de données ORACLE pour qu'il y remédie.

- *La syntaxe est conforme*

  Pour que la syntaxe de la localisation d'un équipement du réseau soit conforme, elle doit se présenter exactement comme suit:

  - le numéro du bâtiment suivi d'un caractère blanc;

  - le numéro de la salle précédé d'un chiffre ou d'une lettre indiquant le numéro de l'étage auquel se trouve la salle (R = Rez-de-chaussée, S = Sous-sol). Un trait d'union sépare les deux numéros;

  - Optionnellement, un nom de salle peut suivre. Il sera toujours précédé par le caractère ":".

    exemple : 513 R-050 : NETWORKS

- *La syntaxe est non conforme*

  Dans le cas où la syntaxe de la localisation est non conforme au format indiqué ci-dessus, l'équipement et sa localisation sont écrits dans un fichier "*Problemes.log*" qui reprend tous les problèmes qui sont survenus durant le traitement. Ceci afin que la personne responsable puisse corriger toutes les localisations possédant une mauvaise syntaxe dans la base de données ORACLE.

2.1.2. L'initialisation de l'attribut "Location"

Si après avoir analysé la syntaxe de la localisation d'un équipement physique celle-ci s'avère conforme, on la copie dans l'attribut "*Location*" de la vue "*Information*" de l'équipement en question par l'entremise du "*SpectroSERVER Application Program Interface*". Ainsi, lorsqu'un utilisateur ouvre une vue pour obtenir les informations sur un équipement, sa localisation apparaît dans le champ "*Location*". La figure 6.1. schématise d'une manière simplifiée cette première partie du projet.

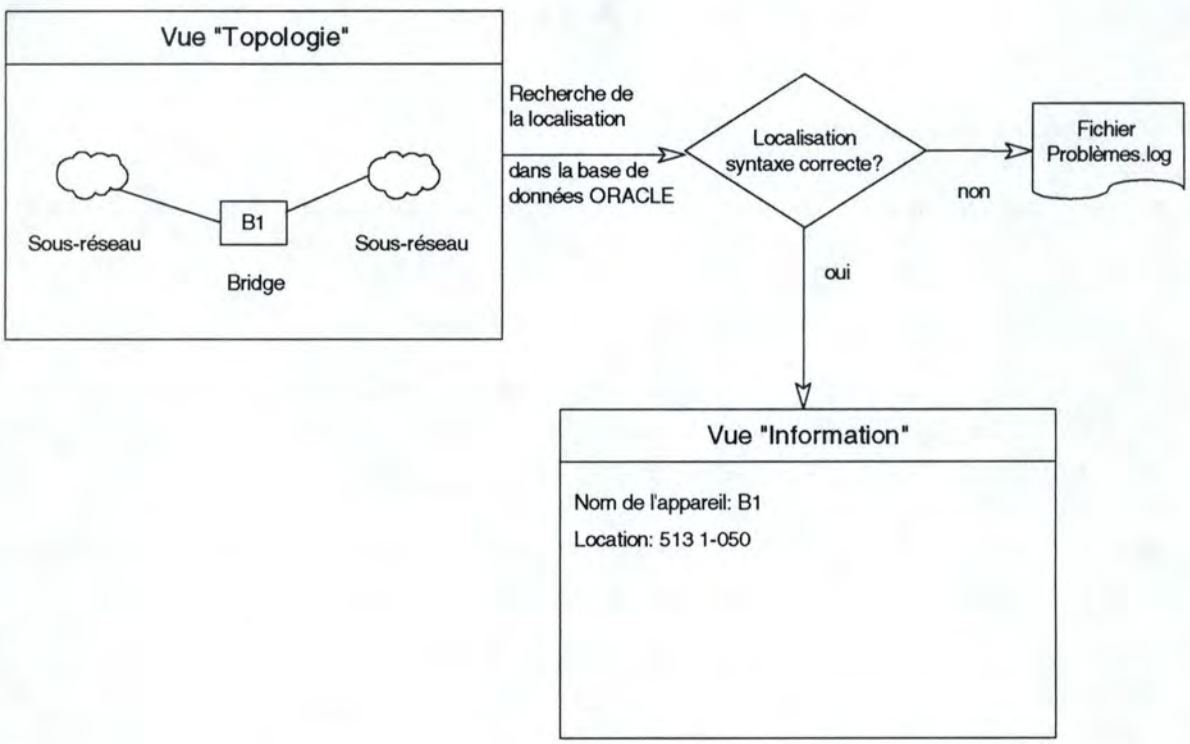


Figure 6.1. Schéma illustrant la première partie du projet "Location_View".

2.2. Deuxième partie

Dans cette deuxième partie, nous devons nous charger de copier les icônes se trouvant dans la vue "*Topologie*" et symbolisant des équipements du réseau vers les vues *"Localisation"* adéquates.

En effet, maintenant que nous disposons de la localisation d'un équipement en termes de bâtiment et, dans beaucoup de cas, de salle, il s'agit de vérifier si le bâtiment (et la salle) existe(nt) dans la base de données de SPECTRUM.

Dans le cas où le bâtiment et la salle sont présents, on copie l'icône symbolisant l'équipement de la vue "*Topologie*" vers la vue *"Localisation"* de la salle adéquate. Etant donné que, pour le moment en tout cas, toutes les salles du CERN contenues dans la base de données de SPECTRUM ont un numéro (ou un nom) unique, une salle dont on dispose du numéro (ou du nom) ne peut appartenir qu'à un seul bâtiment. Comme nous pouvons également supposer que les opérateurs ont pris la précaution de créer dans le logiciel SPECTRUM les salles à l'intérieur des bâtiments adéquats, nous ne devons pas nous occuper de vérifier si la salle dans laquelle on copie une icône appartient bien au même bâtiment figurant dans la localisation obtenue.

Par contre, dans le cas où seul le bâtiment existe dans la base de données de SPECTRUM, on copie l'icône symbolisant l'équipement dans la vue *"Localisation"* qui correspond au bâtiment trouvé.

Ces copies peuvent s'effectuer de deux manières. Soit par l'intermédiaire du SpectroGRAPH par un simple "Copier-Coller" de l'icône, soit par un programme qui copiera tous les équipements voulus dans les vues *"Localisation"* par l'entremise du "*Command Line Interface*" (CLI) et du "*SpectroSERVER Application Program Interface*" (SSAPI).

2.2.1. Le "Copier-Coller" du SpectroGRAPH

La méthode du "Copier-Coller" fait appel à différentes fonctions qu'offre le SpectroGRAPH. Ainsi l'opérateur doit avant tout ouvrir la vue "*Topologie*" dans laquelle se trouve l'icône à copier et la vue *"Localisation"* dans laquelle on copiera l'icône représentant l'équipement. Comme l'ouverture d'une vue ne prend qu'une partie de l'écran, l'opérateur peut visualiser sans problème plusieurs vues en même temps.
Ensuite, il lui suffit de réaliser un simple "Copier-Coller" de l'icône grâce à l'utilisation de la souris et des menus de commandes offerts par le logiciel.

Cette première méthode présente l'avantage de pouvoir placer la copie de l'icône à l'endroit désiré dans la vue *"Localisation"* du bâtiment ou de la salle (par exemple, en haut à gauche dans la vue). Néanmoins, cette manipulation a également deux désavantages.

D'une part, elle risque d'occuper un temps considérable de l'opérateur, surtout si le nombre d'icônes à copier des vues "*Topologie*" vers les vues *"Localisation"* est élevé. De plus, comme le logiciel est en phase de développement, il occasionne pas mal de problèmes qui entraînent souvent une réinitialisation complète. A ce moment-là, tout le travail effectué par l'opérateur est perdu et doit être recommencé.

D'autre part, cette méthode demande une intervention humaine et n'est donc pas automatisable. Par conséquent, on ne peut pas la lancer comme on l'effectuerait pour l'exécution d'un programme.

2.2.2. La copie automatique des icônes par un programme

La seconde méthode consiste à écrire un programme qui copiera automatiquement, c'est-à-dire sans intervention humaine durant son exécution, toutes les icônes représentant des équipements des vues "*Topologie*" vers les vues *"Localisation"* adéquates.

Pour ce faire, ce programme fera appel à deux modules que le logiciel SPECTRUM met à disposition pour le développement de nouvelles applications. Il s'agit du module qui gère le "*Command Line Interface*" (CLI) et du module qui s'occupe de la gestion du "*SpectroSERVER Application Program Interface*" (SSAPI). Le CLI et le SSAPI constituent des interfaces qui offrent un langage permettant d'interroger directement la base de données de SPECTRUM sans devoir passer par l'entremise du SpectroGRAPH qui, lui-même, utilise l'interface SSAPI pour communiquer avec le SpectroSERVER ([SPEC2] & [SPEC4]).

Les avantages de cette seconde méthode résident dans la rapidité et dans la facilité d'exécution du programme. En effet, l'opérateur n'a plus qu'à lancer le programme chaque fois qu'il souhaite configurer ses vues *"Localisation"*.

Mais nous avons également rencontré un problème. En effet, lorsqu'on utilise le CLI et le SSAPI pour copier plusieurs icônes symbolisant des équipements dans une même vue *"Localisation"*, celles-ci se superposent au même endroit dans la vue. Ainsi, lorsque l'utilisateur ouvre une vue *"Localisation"* contenant des icônes, ces dernières se trouvent toutes dans le coin supérieur gauche de la vue.
C'est ensuite à l'opérateur d'aller positionner manuellement, au moyen de la commande "Déplacer" apparaissant dans un menu du SpectroGRAPH, toutes les icônes appartenant à une même vue. On peut facilement imaginer la lourdeur de cette manipulation, surtout lorsqu'on a conscience que l'objectif premier du programme était d'être automatisable et ainsi d'éviter au maximum toute intervention humaine.

Après quelques recherches, nous avons trouvé un moyen pour résoudre ce problème. En effet, lorsqu'une icône est copiée dans une vue *"Localisation"*, on peut indiquer au SpectroSERVER les coordonnées de sa position dans la vue. La valeur de cette position est contenue dans un attribut interne à SPECTRUM prévu à cet effet. Cet attribut appartient à l'ensemble des attributs servant à décrire un modèle.

2.2.3. La méthode choisie

A la vue des avantages et des inconvénients proposés par ces deux méthodes ainsi que des solutions à ces inconvénients, nous avons décidé d'opter pour la copie automatique des icônes par un programme.

Mais avant de lancer la copie automatique, il convient de bien faire attention à ce que toutes les vues *"Localisation"* soient vides lors du lancement du programme pour éviter qu'une icône représentant un équipement soit représentée plusieurs fois dans une même vue *"Localisation"*.
On pourrait toutefois penser que le logiciel SPECTRUM devrait être en mesure d'effectuer lui-même cette vérification avant de placer une icône dans la vue *"Localisation"*, mais la version actuelle ne le propose pas. Nous pouvons seulement espérer que les versions ultérieures vérifieront cette contrainte.

C'est pour cette raison que nous avons également décidé de concevoir un programme qui efface les icônes de toutes les vues *"Localisation"*. Ce programme devrait, pour bien faire, être lancé avant toute exécution automatique de la copie des icônes.
Nous en avons également profité pour écrire un programme supplémentaire qui réinitialise l'attribut "*Location*" des vues "*Information*" des équipements. En réalité, il les remet tout simplement à "blanc". La figure 6.2. schématise d'une manière simplifiée la deuxième partie du projet.

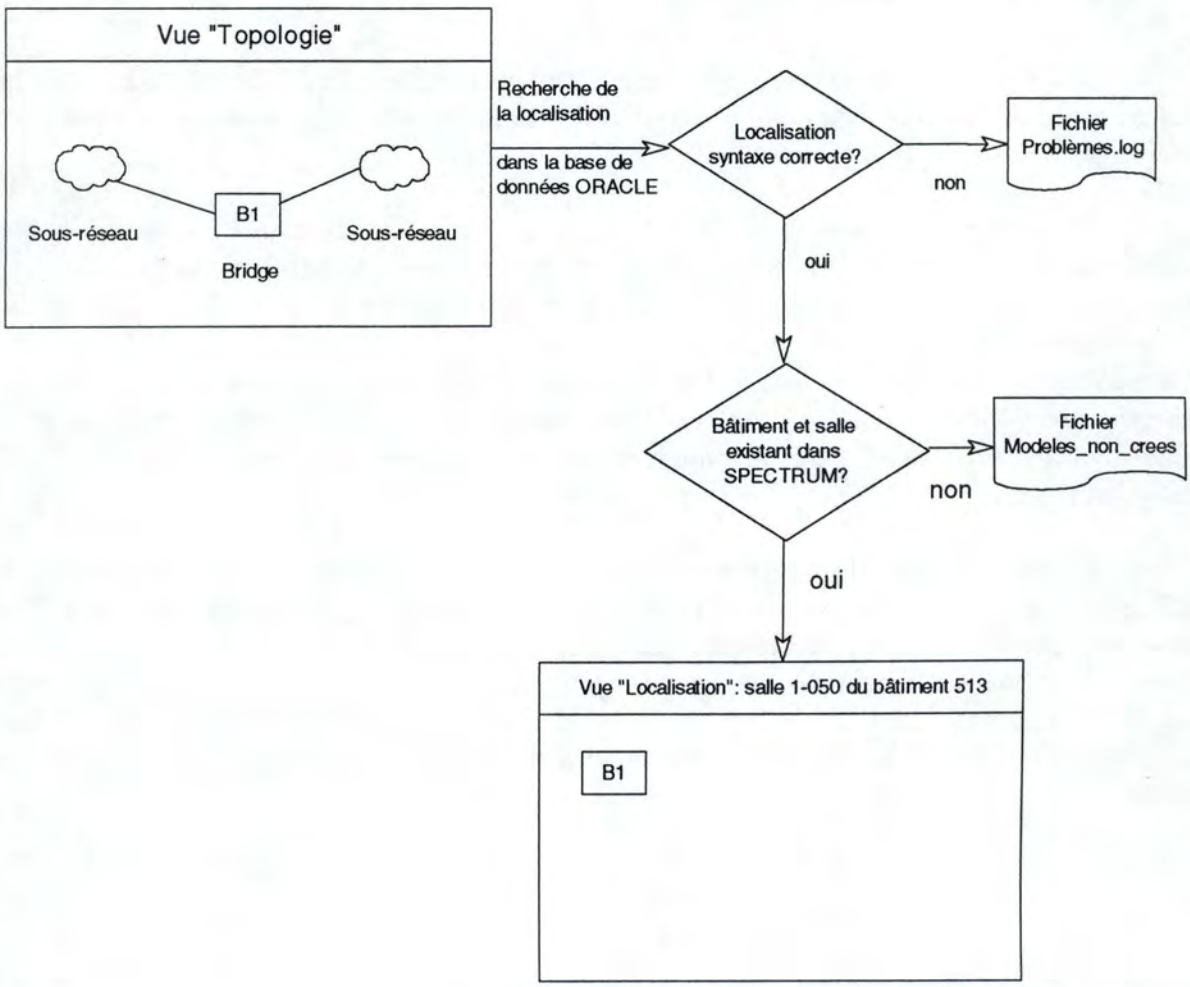


Figure 6.2. Schéma illustrant la deuxième partie du projet "Location_View".

3. La solution implémentée

Maintenant que nous avons expliqué la méthode utilisée pour réaliser notre projet et la motivation de notre choix, nous allons passer à la mise en oeuvre de la solution.

Quatre premiers programmes ont été écrits. Il s'agit des programmes:

1- *Crea_Attr_Location_dans_Spectrum_1*;

2- *Crea_Attr_Location_dans_Spectrum_2*;

3- *Creation_Vues_Localisation_1*;

4- *Creation_Vues_Localisation_2*.

Et comme nous venons de le voir au point 2.2.3., nous avons écrit deux programmes supplémentaires qui se chargent de réinitialiser d'une part les attributs "*Location*" des vues "*Information*" de tous les équipements et d'autre part toutes les vues *"Localisation"*. Ces deux programmes sont:

5- *Supp_Attr_Location_dans Spectrum*;

6- *Suppression_Devices_Des_Vues_Localisation.*

Il est intéressant de faire remarquer que les programmes 1- & 2- (et on peut émettre la même remarque pour les programmes 3- & 4-) doivent être exécutés l'un à la suite de l'autre, puisqu'ils constituent en fait un tout.

La raison pour laquelle ces programmes n'ont pas été fusionnés est qu'ils représentent une combinaison des langages C et C++, accompagnée de requêtes SQL. Et la version du compilateur C++ utilisée lors de notre stage n'acceptait pas les requêtes SQL, rendant impossible la compilation des deux programmes fusionnés.

Pour éviter de devoir lancer d'abord le premier programme et ensuite attendre la fin de son exécution pour pouvoir lancer le deuxième programme, il suffit de les faire exécuter par un "shell-script". De cette manière, l'opérateur n'a plus qu'à lancer une seule chose, le "shell-script".

4. Explication des programmes

Dans cette section, nous allons détailler la manière dont ont été réalisés les différents programmes. Comme nous venons de le mentionner, six programmes ont été conçus.

4.1. Crea_Attr_Location_dans_Spectrum_1

Le programme "*Crea_Attr_Location_dans_Spectrum_1*" crée tout d'abord, par l'entremise du "*Command Line Interface*", une table en mémoire centrale "*Dev_Tbl*" qui reprend tous les équipements répertoriés dans la base de données de SPECTRUM et possédant une adresse IP. Cette adresse IP servira de clé d'accès pour rechercher dans la base de données ORACLE la localisation de l'équipement qu'elle identifie.
Tous les équipements qui ont une adresse IP n'existant pas dans la base de données ORACLE, qui n'ont pas de localisation définie dans cette même base de données ou qui ne possèdent tout simplement pas d'adresse IP dans la base de données de SPECTRUM, sont écrits dans le fichier "*Problemes.log*" destiné aux administrateurs de SPECTRUM et de la base de données ORACLE.

Ensuite, pour chaque équipement se trouvant dans la table "*Dev_Tbl*", on recherche sa localisation par l'intermédiaire d'une requête SQL avec son adresse IP comme clé d'accès. Une fois la localisation trouvée, on l'écrit dans le fichier "*Modeles_Avec_Attr_Location*" précédée par l'identifiant de l'équipement dans SPECTRUM.
Ce fichier "*Modeles_Avec_Attr_Location*" servira de fichier d'entrée au programme "*Crea_Attr_Location_dans_Spectrum_2*".

4.2. Crea_Attr_Location_dans_Spectrum_2

Comme nous l'avons précisé dans le point 3., le programme "*Crea_Attr_Location_dans_Spectrum_2*" doit être exécuté à la suite du programme "*Crea_Attr_Location_dans_Spectrum_1*" puisque, pour pouvoir être lancé et fournir des résultats corrects, il a besoin du fichier "*Modeles_Avec_Attr_Location*" créé par ce dernier programme.

Pour rappel, chaque ligne de ce fichier reprend l'identifiant interne à SPECTRUM d'un équipement ainsi que sa localisation. Par conséquent, pour chaque identifiant d'équipement, le programme va lire sa localisation et s'en servir pour initialiser l'attribut "*Location*" de la vue "*Information*" de l'équipement.

A cette fin, il utilise le "*SpectroSERVER Application Program Interface*" pour interagir directement avec la base de données de SPECTRUM, sans passer par le "*Command Line Interface*" qui, lui, ne permet pas de modifier les attributs "*Location*".

4.3. Creation_Vues_Localisation_1

Le programme "*Creation_Vues_Localisation_1*" s'occupe dans un premier temps de créer trois tables en mémoire centrale:

- la table "*Building_Tbl*" qui reprend tous les bâtiments recensés dans la base de données de SPECTRUM. Ils sont accompagnés de leur identifiant interne au logiciel;

- la table "*Room_Tbl*" qui reprend toutes les salles recensées dans la base de données de SPECTRUM. Elles sont accompagnées de leur identifiant interne et du nombre d'équipements qu'elles contiennent. Ce nombre servira lors du positionnement des icônes représentant des équipements dans les vues *"Localisation"*;

- la table "*Dev_Tbl*" qui reprend tous les équipements supportant le protocole SNMP ("GnSNMPDev"), les "pingables" et les routeurs ("Rtr_Shiva") recensés dans la base de données de SPECTRUM et qui possèdent une adresse IP. Ils sont accompagnés de leur identifiant interne, de leur adresse IP et de leur type. (Remarque: tous les

équipements ne possédant pas d'adresse IP sont écrits dans le fichier "modeles_non_crees")

Ensuite, pour chaque équipement se trouvant dans la table "*Dev_Tbl*", on recherche sa localisation au moyen d'une requête SQL avec comme clé d'accès son adresse IP.

Si la syntaxe de la localisation n'est pas conforme au format décrit au point 2.1.1., l'équipement n'est placé dans aucune des vues *"Localisation"*. Le nom de l'équipement et son identifiant interne sont alors écrits dans le fichier "modeles_non_crees" pour en avertir l'administrateur de SPECTRUM.

Après avoir trouvé une localisation avec une syntaxe conforme, cette localisation va être analysée par le programme pour en extraire le numéro du bâtiment et, s'il existe, le numéro (ou le nom) de la salle qui contiennent l'équipement.

L'analyse consiste à vérifier si le bâtiment dans lequel l'équipement se situe existe dans la base de données de SPECTRUM. Cette vérification s'effectue en comparant le numéro du bâtiment trouvé dans la localisation et tous les numéros de bâtiment recensés dans SPECTRUM et qui sont repris dans la table "*Building_Tbl*". Comme le nombre de bâtiments du CERN n'est pas très élevé, il nous a semblé inutile d'arranger par ordre croissant les numéros des bâtiments dans la table "*Building_Tbl*", ce qui aurait permis une recherche dichotomique sur les numéros de bâtiment.

Si le bâtiment existe et que rien n'est dit sur la salle particulière à laquelle l'équipement en question appartient, le programme place l'icône représentant l'équipement dans la vue *"Localisation"* de ce bâtiment. Il indique en plus dans le fichier "*modeles_crees_dans_buildings*" que l'équipement a bien été copié et dans quel bâtiment. A ces deux informations vient s'ajouter l'identifiant interne de l'équipement.

Par contre, si la bâtiment n'est pas répertorié dans la base de données de SPECTRUM, le programme écrit dans le fichier "modeles_non_crees" l'identifiant de l'équipement, son nom et le numéro du bâtiment.

Dans le cas où le bâtiment existe et que l'équipement appartient aussi à une salle particulière, le programme s'occupe de vérifier si cette salle est recensée dans la base de données de SPECTRUM. Pour cela, il réalise une nouvelle comparaison entre le numéro (ou le nom) de la salle figurant dans la localisation et celui de l'ensemble des salles existantes dans la table "*Room_Tbl*".
Si la salle existe bien dans la base de données du logiciel, le programme place l'icône représentant l'équipement, non pas dans la vue *"Localisation"* correspondant au bâtiment, mais bien dans celle de la salle.
De plus, le programme indique dans le fichier "*modeles_crees_dans_rooms*" tous les équipements qui ont été placés dans des salles et incrémente d'une unité le nombre d'équipements que contient la salle.

Par contre, si la salle n'est pas répertoriée dans la base de données de SPECTRUM, l'icône modélisant l'équipement est copiée dans la vue *"Localisation"* correspondant au bâtiment dans lequel se trouve la salle. Le numéro (ou le nom) de salle et l'identifiant de l'équipement sont écrits dans le fichier "modeles_non_crees".

En plus, lorsqu'un équipement est placé dans une vue *"Localisation"*, que ce soit celle d'un bâtiment ou celle d'une salle, le programme l'indique dans le fichier "*modeles_crees_dans_spectrum*".

Nous nous permettons de rappeler que dans le cas où la localisation de l'équipement contient le nom d'une salle (":nom_de_la_salle"), le programme privilégie ce nom par rapport au numéro de cette même salle. On procède de cette manière car dans un futur plus ou moins proche, certains numéros de salle du CERN seront amenés à être remplacés par des noms de salle plus symboliques.

Et finalement, le programme termine son exécution par la création du fichier "*Rooms*" à partir de la table "*Room_Tbl*".

Pour essayer de clarifier la situation, nous allons reprendre ci-dessous une description détaillée des six fichiers créés tout au long de l'exécution du programme:

- "*modeles_crees_dans_spectrum*" : ce fichier reprend l'ensemble des équipements qui ont été copiés dans les vues *"Localisation"* adéquates.

  Son format est le suivant :

  - l'identifiant interne (sous forme hexadécimale) du bâtiment ou de la salle où a été placé l'équipement;
  - l'identifiant interne (sous forme hexadécimale) de l'équipement;
  - le nom de l'équipement;
  - le nom du bâtiment ou le numéro (ou le nom) de la salle.

- "*modeles_crees_dans_buildings*" : ce fichier reprend l'ensemble des équipements qui ont été copiés dans les vues *"Localisation"* correspondant au bâtiment adéquat.

Son format est le suivant :

- l'identifiant interne (sous forme hexadécimale) du bâtiment dans lequel se trouve l'équipement;
- l'identifiant interne (sous forme hexadécimale) de l'équipement;
- le type de l'équipement;
- le nom de l'équipement;
- le nom du bâtiment.

- "*Modeles_crees_dans_rooms*" : ce fichier reprend l'ensemble des équipements qui ont été copiés dans les vues *"Localisation"* correspondant à des salles.

  Son format est le suivant :

  - l'identifiant interne (sous forme hexadécimale) de la salle dans laquelle se trouve l'équipement;
  - l'identifiant interne (sous forme hexadécimale) de l'équipement;
  - le type de l'équipement;
  - le nom de l'équipement;
  - le numéro (ou le nom) de la salle.

- "*Rooms*" : ce fichier reprend, pour chaque salle contenant au moins un équipement, l'identifiant interne de la salle et le nombre exact d'équipements qu'elle contient.

- "modeles_non_crees": ce fichier reprend l'ensemble des équipements qui n'ont pas été copiés dans une des vues *"Localisation"*, et le bâtiment ou la salle dans lequel ils auraient dû être copiés et n'existant pas actuellement dans la base de données de SPECTRUM.
  Il reprend également les localisations présentant une mauvaise syntaxe.

Le fichier "*modeles_crees_dans_spectrum*" servira comme fichier d'entrée au programme "Suppression_Modeles_Des_Vues_Localisation".

Les fichiers "*Rooms*" et "*modeles_crees_dans_rooms*" serviront comme fichiers d'entrée au programme "*Creation_Vues_Localisations_2*".
Les fichiers "*modeles_crees_dans_buildings*" et "modeles_non_crees" sont créés à titre informatif et destinés aux différents opérateurs et administrateurs.

4.4. Creation_Vues_Localisation_2

Le programme "*Creation_Vues_Localisation_2*" se charge du positionnement des équipements, qui sont contenus dans le fichier "modeles_crees_dans_room", dans les vues *"Localisation*" des salles adéquates.

A partir du fichier "*Rooms*" créé par le programme "*Creation_Vues_Localisation_1*", on connaît le nombre d'équipements que contient une salle. Nous avons vu également qu'il était possible de positionner une icône à un endroit précis dans une vue *"Localisation*". Ainsi, le programme "*Creation_Vues_Localisation_2*" peut donc disposer les icônes dans la vue de manière à ce qu'elles ne se superposent pas.

Ce positionnement dépend de deux paramètres:

- La position en "X" voulue de l'équipement (en pixels);
- La position en "Y" voulue de l'équipement (en pixels);

Le programme utilise quatre procédures:

- **atoX** : elle transforme un string représentant un entier sous forme hexadécimale en un string sous forme décimale;
- **inverse** : elle inverse un string par tranche de deux bits qu'elle sépare par des points (exemple: 4008ff -> ff.8.40.0);
- **calcul_positionX** : elle calcule la position en "X" d'un équipement dans une vue *"Localisation*" en fonction du nombre d'équipements devant figurer dans la vue;
- **calcul_positionY** : elle calcule la position en "Y" d'un équipement dans une vue *"Localisation*" en fonction du nombre d'équipements devant figurer dans la vue.

Dans un premier temps grâce à ces quatre procédures, le programme construit, bit par bit, la valeur de l'attribut d'une vue *"Localisation*" qui caractérise la position de l'ensemble des icônes y figurant.
Cet attribut commence par un nombre sous forme hexadécimale qui indique le nombre d'équipements contenus dans la vue *"Localisation*". Il est ensuite composé d'autant de suites de

cinq nombres sous forme hexadécimale qu'il existe d'équipements dans la vue. Voici ce que représentent ces différents nombres:

- le 1er nombre représente l'identifiant de l'équipement dans la base de données de SPECTRUM;
- le 2e nombre représente le type de l'équipement;
- le 3e nombre a toujours une valeur égale à 0;
- le 4e nombre représente la position en "X" de l'équipement (en pixels);
- le 5e nombre représente la position en "Y" de l'équipement (en pixels);

Dans un deuxième temps, le programme initialise l'attribut à cette valeur grâce au SSAPI. Et il réitère ce processus pour toutes les salles contenant au moins un équipement.

4.5. Supp_Attr_Location_dans_Spectrum

Le programme "Supp_Attr_Location_dans Spectrum" efface la valeur de l'attribut "*Location*" qui correspond à la localisation d'un équipement dans sa vue "*Information*", et ce, pour tous les équipements se trouvant dans le fichier "*Modeles_Avec_Attr_Location*" créé par le programme "*Crea_Attr_Location_dans_Spectrum_1*".
De cette manière, lorsqu'on ouvre la vue "*Information*" d'un équipement, le champ "*Location*" est vide.

Durant son exécution, le programme utilise le SSAPI pour "mettre à blanc" l'attribut "*Location*" ainsi que la même procédure "atoX" précédemment décrite.

4.6. Suppression_Devices_Des_Vues_Localisation

A partir du fichier "*modeles_crees_dans_spectrum*" qui a été créé par le programme "*Creation_Vues_Localisation_1*", ce programme-ci efface toutes les icônes représentant des équipements qui ont été placées dans les vues *"Localisation*".
Cet effacement se réalise par l'intermédiaire du "*Command Line Interface*".

Remarque :

Pour stocker les différents éléments de SPECTRUM tels que les équipements, les bâtiments et les salles durant l'exécution des programmes, deux possibilités nous étaient offertes: nous pouvions utiliser soit de simples fichiers, soit des tables situées en mémoire centrale.

Finalement, pour une question de rapidité, nous avons opté pour l'utilisation de tables en mémoire centrale. En effet, vu que le nombre d'accès à toutes ces données est très élevé, l'utilisation de fichiers entraîne des temps d'accès non négligeables pénalisant ainsi le temps d'exécution des programmes.
Certes, la création de ces tables au début de chaque programme par l'intermédiaire du "*Command Line Interface*" exige également un certain temps. Mais ce temps est beaucoup moins important que celui exigé par l'utilisation des fichiers.
Le seul véritable problème avec l'utilisation des tables en mémoire centrale, c'est que leur longueur est fixée et donc limitée à priori. Il convient d'estimer à l'avance, c'est-à-dire dans le code source des programmes, la longueur maximale de ces tables (contenue dans une constante).

Par conséquent, pour éviter toute mauvaise surprise dûe à une longueur de table trop petite, le programme réalise une comparaison entre le nombre d'éléments que devrait contenir une table et la longueur effective de la table. Si ce nombre d'éléments s'avére trop grand, l'utilisateur en est averti ainsi que de la nouvelle longueur à attribuer à la table. Il ne lui reste plus qu'à modifier la valeur de la constante dans le code source et ensuite à recompiler ce dernier.

5. La phase de test des programmes

Pour réaliser le test des programmes, nous avons dû effectuer beaucoup de manipulations avec le SpectroGRAPH telles que de nombreuses ouvertures et fermetures de vues et formuler des requêtes SQL pour interroger la base de données ORACLE. Ainsi, nous avions la possibilité de vérifier si la valeur de l'attribut "*Location*" de la vue "*Information*" d'un équipement et la vue *"Localisation*" dans laquelle l'icône le symbolisant était copiée correspondaient bien à la localisation de l'équipement en question trouvée dans la base de données ORACLE.

De même, nous devions nous assurer qu'une icône symbolisant un équipement qui n'était pas copiée dans une des vues *"Localisation*" l'était pour une raison valable comme par exemple un bâtiment ou une salle non répertorié dans SPECTRUM.

Pour terminer, le positionnement des icônes dans une vue "*Localisation*" nous a également pris un certain temps. En effet, nous devions "calibrer", par l'intermédiaire d'essais répétés, les petites procédures qui calculent la position des icônes dans une vue "*Localisation*" en fonction de leur nombre.

6. Conclusion

Ce sixième chapitre met fin à la présentation des trois projets que nous avons développés au CERN durant nos six mois de stage.
Au contraire de nos deux premiers projets "LectureDonnees" et "Alarm_Mirror" qui constituent des projets critiques dont la réussite conditionne fortement la mise en production du logiciel SPECTRUM, le projet que nous venons de présenter dans ce chapitre n'était pas d'une importance capitale. Il s'agit seulement d'une facilité fournie aux opérateurs et à l'administrateur du réseau pour leur permettre de localiser aisément un équipement du réseau. Néanmoins, ce troisième projet nous aura pris un temps non négligeable, surtout en ce qui concerne le problème du positionnement des icônes dans les vues "*Localisation*". Nous pouvons toutefois espérer que les versions ultérieures du logiciel SPECTRUM permettront un positionnement automatique des icônes dans ces vues.

Nous nous permettons d'informer le lecteur que, s'il désire consulter les codes sources des six programmes constituant le projet "Location_View", il peut les trouver à l'annexe 5 située à la fin de ce mémoire.

Chapitre 7:
Deux autres logiciels de gestion de réseaux

Dans ce dernier chapitre, nous allons présenter les grandes lignes de deux autres logiciels de gestion de réseaux qui se trouvent actuellement sur le marché et qui rencontrent un certain succès auprès des administrateurs et opérateurs de réseaux. Il s'agit des logiciels OPENVIEW de Hewlett Packard et NETVIEW de I.B.M. qui ont tous deux été conçus et développés aux Etats-Unis.
Durant cette présentation, nous tenterons de réaliser des comparaisons avec le logiciel SPECTRUM chaque fois qu'il nous en semblera bon.

1. Le logiciel HP Openview Network Management

Le logiciel de gestion de réseaux OPENVIEW a été développé par Hewlett-Packard Company. Toutes les informations figurant dans cette présentation de OPENVIEW ont été tirées de documents fournis par la société Hewlett-Packard à Bruxelles dont les références bibliographiques sont [HP1] et [HP2].

Construit sur base de standards de l'industrie, OPENVIEW permet de réagir avec rapidité aux événements quotidiens pouvant survenir sur un réseau. Il a également la possibilité de s'adapter facilement aux modifications ultérieures du réseau qu'il gère et possède l'avantage de pouvoir tourner aussi bien dans les environnements LAN que dans les environnements WAN (Wide Area Network).
Nous allons maintenant développer quelques-unes de ses caractéristiques.

a) La gestion de réseaux intégrés

Le logiciel OPENVIEW offre diverses fonctions pour contrôler les composants d'un réseau ainsi que son flux de trafic. Toutes ces fonctions permettent aux administrateurs et opérateurs du réseau de déceler rapidement tout problème au niveau d'un composant.
Dans le logiciel SPECTRUM, nous retrouvons ce rôle au niveau du SpectroSERVER qui offre une myriade de facilités aux administrateurs et opérateurs.

Disponible sous Windows et les plates-formes HP-UX, OPENVIEW est capable de gérer des réseaux intégrés de toutes tailles, allant du plus petit LAN au plus grand WAN.

b) Un support multi-plate-formes

Le logiciel OPENVIEW est disponible sur une plate-forme PC, tournant sous Windows ou Windows NT, pour la gestion des réseaux de petite ou de moyenne taille. Pour les réseaux plus étendus, et les applications exigeant une grande puissance de calcul, OPENVIEW est disponible sur des stations de travail HP et tourne sous le système d'exploitation UNIX RISC.

c) Le support de standards

En utilisant le standard SNMP comme protocole de gestion tout comme le logiciel SPECTRUM, OPENVIEW peut communiquer avec un grand nombre d'équipements qui implémentent de plus en plus ce protocole. Le logiciel supporte les objets standards de la MIB-II ainsi que les extensions propriétaires de la MIB proposées par certains vendeurs.
OPENVIEW propose également une fonction "Auto-Discovery" semblable à celle offerte dans SPECTRUM qui discerne automatiquement tous les équipements à gérer sur le réseau et les insère de façon précise dans son plan reflètant la topologie du réseau. C'est sur ce plan que se basent les applications de OPENVIEW s'occupant de la gestion des équipements. Il suffit à l'utilisateur d'aller cliquer sur une icône présente dans la topologie et représentant un équipement pour gérer ce dernier et en obtenir des renseignements divers. Ce mécanisme est aussi d'application avec le logiciel SPECTRUM qui offre des vues "*Information*" pour obtenir des renseignements sur un équipement.

d) La facilité d'utilisation

Grâce aux facilités d'utilisation du logiciel OPENVIEW, l'utilisateur peut se consacrer à la gestion de son réseau plutôt que de passer son temps à se tracasser des différentes représentations graphiques de la topologie du réseau ou encore de la multitude de caractéristiques des équipements.
Un autre avantage du logiciel OPENVIEW à côté de ses fonctions attrayantes se situe au niveau de la convivialité de ses interfaces graphiques. Lorsque l'utilisateur désire sélectionner un

équipement, il suffit de pointer sur ce dernier avec la souris et de cliquer dessus. OPENVIEW dispose également d'une aide et d'une documentation "On-Line".
Ici aussi, au niveau de la convivialité et de l'utilisation des interfaces graphiques, nous pouvons faire un rapprochement entre SPECTRUM et OPENVIEW.

e) La gestion du trafic à travers le réseau

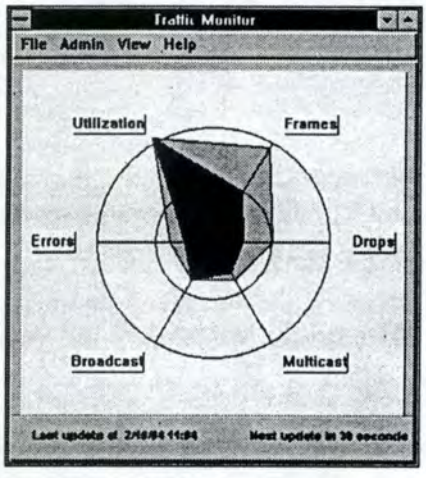
Les applications offertes par le logiciel OPENVIEW pour gérer le trafic telles que "Traffic Monitor", "History Analyser", et "Traffic Expert" sont basées sur le HP EASE (Embedded Advanced Sampling Environment"), un essai de technologie innovatrice qui est intégrée dans chaque composant HP géré dans le réseau (hub, bridge, router...). Ces applications fournissent des analyses simplifiées, des planifications et des contrôles du flux du trafic dans le réseau. Ainsi, l'utilisateur peut analyser le trafic passant sur tous les segments du réseau et sur les WANs auxquels ils sont connectés.
A notre avis, les analyses, les planifications et les contrôles du trafic réalisés par les applications de OPENVIEW peuvent s'apparenter au "Statistic Manager" et au "Event Manager" du logiciel SPECTRUM.

Quelques caractéristiques de l'application "Traffic Monitor"

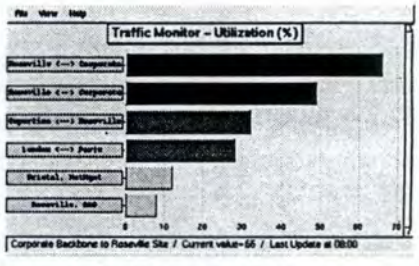
- L'application "Traffic Monitor" constitue un contrôleur de trafic fonctionnant en temps réel dans un environnement LAN/WAN et qui fournit une vue sur le trafic d'un réseau au sein d'une entreprise. Elle calcule et enregistre en temps réel les fluctuations du trafic sur le réseau, indique à l'utilisateur les points "chauds" du réseau et l'avertit de toute occurrence d'un problème sur le réseau.

- L'application "Traffic Monitor" fournit un processus contrôle composé de trois étapes.
  Premièrement, l'utilisateur peut observer les problèmes qui existent en parcourant la vue "*Entreprise*" (vue équivalente à la vue "*Topologie*" de SPECTRUM) fournie par le diagramme "Radar". Ce diagramme affiche les paramètres les plus importants qui renseignent sur la santé du réseau.
  Deuxièmement, après avoir procédé à l'identification des paramètres possédant des valeurs anormales, "Traffic Monitor" aide l'utilisateur à localiser le problème. Les segments LAN ou les connexions des routeurs qui occasionnent le problème sont listés par ordre décroissant d'importance.
  Et troisièmement, l'application permet à l'utilisateur de savoir de quel composant provient le problème: du noeud source, du noeud destination ou bien des deux en même temps.

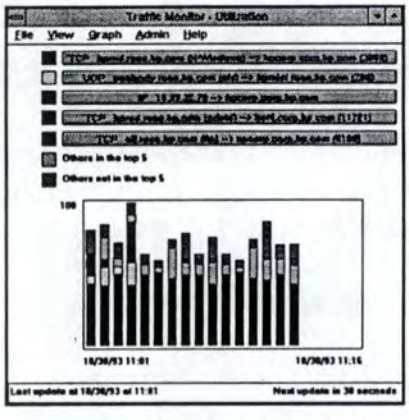
Les fenêtres correspondant aux trois étapes sont illustrées à la figure 7.1.



Radar diagram: What is the problem?



Segment view: Where is the problem?



Node view: Who is causing the problem?

Figure 7.1. Les différentes fenêtres proposées par l'application "Traffic Monitor".

- L'application "Traffic Monitor" permet un contrôle des seuils et des actions à entreprendre en réponse aux alarmes. Elle analyse avec "intelligence" les paramètres clés du réseau (trafic broadcast, trafic multicast, taux d'erreurs, utilisation des liaisons WAN...) et aide l'utilisateur à prévenir les problèmes. L'utilisateur peut indiquer les actions à entreprendre lorsqu'une certaine condition d'alarme est atteinte (par exemple, l'envoi d'un message à une certaine personne). De plus, tout comme dans le logiciel SPECTRUM, des couleurs sont utilisées pour indiquer l'état dans lequel se trouvent les composants du réseau, jaune signifiant un avertissement et rouge signifiant une alerte.

- "Traffic Monitor" donne de l'information précise sur le trafic du réseau plutôt que des masses de données présentées sans structure bien définie. Son utilisation en est ainsi

facilitée. Par exemple, pour descendre d'une vue au niveau de l'entreprise vers une vue isolant un segment particulier avec ses noeuds, cela ne prend seulement que trois "clicks" avec la souris (les trois types de vue sont la vue globale, la vue sur les segments et la vue sur les noeuds). Ce mécanisme est semblable à celui de SPECTRUM qui invite aussi l'utilisateur à cliquer sur une icône de la vue "*Topologie*" symbolisant un sous-réseau pour obtenir de plus amples informations sur ce dernier. Ensuite, s'il le désire, il peut cliquer sur un équipement du sous-réseau pour disposer de la vue "*Information*" donnant des informations plus précises et propres à l'équipement.

L'utilisateur de OPENVIEW ne souhaitant pas un contrôle sur tout le réseau, peut configurer l'application de manière à ne disposer que d'un sous-ensemble des composants. Cette configuration se réalise rapidement par l'intermédiaire d'une interface graphique.

- L'architecture HP EASE, basée sur un échantillonnage statistique, est tout-à-fait conçue pour l'émergence des architectures de LAN à 100 Mbits/s. Au lieu de récolter et analyser l'information contenue dans chaque paquet transitant sur le réseau, l'application "Traffic Monitor" échantillonne intelligemment les paquets pour en extraire une tendance sur la bonne marche du réseau. Par exemple, "Traffic Monitor" regarde, pour un segment particulier, un paquet toutes les x secondes. Et c'est seulement en fonction de l'information contenue dans ce paquet qu'il établit la "santé" du segment. Ce mécanisme pose l'hypothèse qu'un problème ne doit être pris en compte que lorsque plusieurs paquets en témoignent. Si un seul paquet pris dans la masse indique la survenance probable d'un problème, il ne convient pas encore de le signaler à l'utilisateur.
  Par conséquent, la surcharge du réseau due au processus de contrôle n'est pas trop importante.

2. Le logiciel NetView

Le logiciel NETVIEW V2R2 (Version 2, Release 2) a été conçu par la société I.B.M. pour la gestion des réseaux d'entreprise. Il fournit un ensemble d'outils faciles à utiliser pour la gestion de réseaux complexes et fabriqués par de multiples vendeurs. Tous ces outils peuvent être installés indépendamment l'un de l'autre grâce au concept de "package" offert par NETVIEW. Ainsi, l'utilisateur peut n'installer que les fonctions dont il éprouve besoin.
NETVIEW est aussi un système de gestion contrôlable à partir d'un seul point spécifique.

Pour réaliser cette présentation du logiciel NETVIEW, nous nous sommes basés sur un document qui provient de la société I.B.M. située à Bruxelles et dont voici la référence bibliographique [IBM92].

2.1. La gestion du réseau point-à-point

Le logiciel NETVIEW réalise la gestion du réseau au travers de trois concepts de base: des "*focal points*", des "*Entry points*" et des "*Service points*".
Bien qu'un "*Entry point*" aille toujours de pair avec un "*focal point*", ces points ne doivent pas nécessairement faire partie de la même unité physique.

Voici la description de ces trois concepts:

- Un "*focal point*" est un ensemble de produits appartenant à une application de gestion de réseaux centralisée qui s'occupe de la gestion des composants du réseau. Un "*focal point*" peut ainsi gérer les composants en fonction de leurs problèmes, de leurs changements ou du contrôle de leurs opérations. Nous pouvons en fait considérer le "*focal point*" comme le SpectroSERVER de SPECTRUM.

- Un "*Entry point*" est un produit SNA (protocole d'I.B.M.) qui relie les ressources SNA du réseau au "*focal point*". Les données sur la gestion du réseau et les informations sur les erreurs sont envoyées à partir d'un "*Entry point*" vers un "*focal point*".

- Un "*Service point*" constitue un ou plusieurs produits qui relient des composants non SNA du réseau au "*focal point*". Le "*Service point*" est adressable avec le protocole SNA. Il peut donc traduire les données et les demandes exprimées ou non en SNA entre le SNA "*focal point*" et les équipements ne supportant pas SNA. En réalité, nous pensons que le "*Service point*" joue un peu le même rôle que le "proxy-agent" dans le logiciel SPECTRUM.

L'échange de données sur la gestion du réseau se réalise dans les deux sens entre le "*focal point*" et le "*Entry point*", ainsi qu'entre le "*focal point*" et le "*Service point*". Les données échangées d'un "*Entry point*" ou d'un "*Service point*" vers un "*Focal Point*" portent sur l'identité, le statut et les performances des équipements connectés à ces différents points.
A l'opposé, les données en provenance du "*Focal Point*" et dirigées vers un "*Entry point*" ou un "*Service point*" représentent des commandes à l'attention d'équipements spécifiques du réseau afin que ceux-ci exécutent certaines fonctions.

Par conséquent, NETVIEW a l'avantage de fournir toute une gamme d'outils pour gérer une grande variété de réseaux: les réseaux SNA, les réseaux non-SNA et les réseaux multi-vendeurs.

2.2. Un système et une gestion de réseaux centralisés

Un autre avantage du logiciel NETVIEW se situe dans le fait qu'il permet à l'utilisateur de centraliser un système et la gestion du réseau. Cela se réalise avec l'environnement d'une simple machine hôte ou bien en consolidant les opérations de plusieurs systèmes dans un environnement constitué de plusieurs machines hôtes.

Dans un réseau disposant de plusieurs hôtes, NETVIEW peut, à partir d'un hôte central, gérer un système et des opérations sur le réseau pour des hôtes distribués sans intervention humaine. C'est cela qu'on appelle la "consolidation". Seules les opérations nécessitant un opérateur sont envoyées au "*Focal Point*". Ainsi, lorsqu'il y a consolidation, la gestion du réseau demande moins de dextérité de la part des opérateurs et moins de ressources pour supporter les diverses opérations sur le réseau.

2.3. La gestion automatique du réseau et des systèmes

Le logiciel NETVIEW offre à l'utilisateur la possibilité d'automatiser plusieurs systèmes d'exploitation ainsi que plusieurs tâches propres à la gestion de réseaux. De cette manière, la quantité de tâches répétitives et routinières qui doivent normalement être exécutées par les opérateurs dans un environnement avec une ou plusieurs machines hôtes est considérablement diminuée.

Le contrôle des systèmes

Le logiciel NETVIEW permet à l'utilisateur d'interagir avec le système d'exploitation en lui envoyant des commandes et de recevoir des messages en provenance de programmes tournant en tant que sous-systèmes du système d'exploitation. L'utilisateur peut, par exemple, activer ou désactiver un système, mettre son horloge à l'heure ou exécuter toute autre commande spécifique à un périphérique.

Dans un environnement avec plusieurs machines hôtes, l'utilisateur peut automatiser NETVIEW de sorte que les opérations sur le réseau soient automatiquement exécutées dans les diverses machines hôtes. Les événements significatifs qui demandent une intervention sont envoyées vers le "*Focal Point*" où un opérateur pourra y répondre.

Le traitement des événements

Lorsque des événements se passent sur le réseau (par exemple, une ressource tombant en panne), le système et certaines ressources du réseau émettent des

messages ou des alertes. L'utilisateur peut faire appel à un programme proposé par NETVIEW pour filtrer ces messages et ces alertes. En fonction de leur gravité, NETVIEW réagira soit automatiquement sans intervention de l'opérateur, soit en envoyant un message à l'opérateur pour qu'il effectue une analyse plus poussée du message ou de l'alerte avant d'entreprendre toute action.
Les messages n'exigeant pas d'actions peuvent être soit enregistrés dans un fichier, soit détruits.

L'utilisateur du logiciel NETVIEW peut aussi utiliser la commande "Timer" pour lancer l'exécution automatique d'une opération du système de gestion pendant un intervalle de temps spécifié. De ce fait, nous pourrions peut-être comparer cette commande avec la fonction "DATAEXPORT" du logiciel SPECTRUM qui nous permet d'extraire des informations contenues dans sa base de données et se rapportant à un intervalle de temps précis.

2.4. Les opérations et le contrôle sur le système et sur le réseau

Le logiciel NETVIEW propose une large gamme de services et de fonctionnalités spécifiques pour assister les opérateurs lorsqu'ils contrôlent et opèrent sur les systèmes et sur les réseaux.
Ainsi, les opérateurs ont la possibilité de contrôler des systèmes tournant sur une ou plusieurs machines hôtes à partir d'une console sur laquelle se trouve NETVIEW.

Grâce à certaines fonctionnalités proposées par NETVIEW, l'opérateur peut facilement gérer le statut des ressources du réseau. Le logiciel collecte d'abord le statut des différentes ressources soit à partir d'un "*Entry point*" si la ressource supporte SNA, soit à partir d'un "*Service point*" si la ressource ne supporte pas SNA. Ensuite, il affiche les statuts des différentes ressources en utilisant son "Graphic Monitor" qui représente une vue sur les composants du réseau. A partir de cette vue, l'utilisateur peut envoyer des commandes telles qu'activer ou désactiver une ressource. Nous reparlerons plus en détail de l'utilitaire "Graphic Monitor" un peu plus loin dans ce chapitre.

L'accès aux bases de données

L'utilitaire "NetView Bridge" du logiciel NETVIEW offre une connexion entre le système NETVIEW et la ou les bases de données contenant les informations sur la gestion du réseau et des systèmes. De cette manière, l'utilisateur a la possibilité de développer des applications externes qui utilisent cette information et qui sont susceptibles de mettre à jour ces bases de données.
Ici, nous pourrions établir une comparaison entre l'utilitaire "NetView Bridge" et le "*SpectroSERVER Application Program Interface*" du logiciel SPECTRUM.

2.5. La détermination et le diagnostic des problèmes rencontrés sur le réseau

Afin d'aider l'opérateur à contrôler, identifier et fixer les problèmes survenant sur le réseau, NETVIEW possède plusieurs outils pour déterminer et effectuer un diagnostic sur un problème. Les outils les plus importants sont le "Graphic Monitor", le contrôleur "hardware", le contrôleur de statut, le contrôleur de session et le "Command Facility".
Ces outils utilisent une approche hiérarchique pour aider l'opérateur à isoler un problème.

a) Le "Graphic Monitor"

Le "Graphic Monitor" est une interface graphique permettant de contrôler le statut des ressources du réseau au niveau local, au niveau d'un pays ou au niveau mondial (voir figure 7.2.). "Graphic Monitor" permet aussi d'isoler rapidement les ressources occasionnant une panne. Cette interface représente une vue sur les équipements du réseau qui sont représentés par des icônes dont la couleur dépend du statut de l'équipement. Nous pourrions sans doute assimiler l'utilitaire "Graphic Monitor" aux vues "*Topologie*" et "*Localisation*" de SPECTRUM.
L'opérateur a également la possibilité d'émettre des commandes vers les ressources à partir de l'interface pour résoudre certains problèmes du réseau.

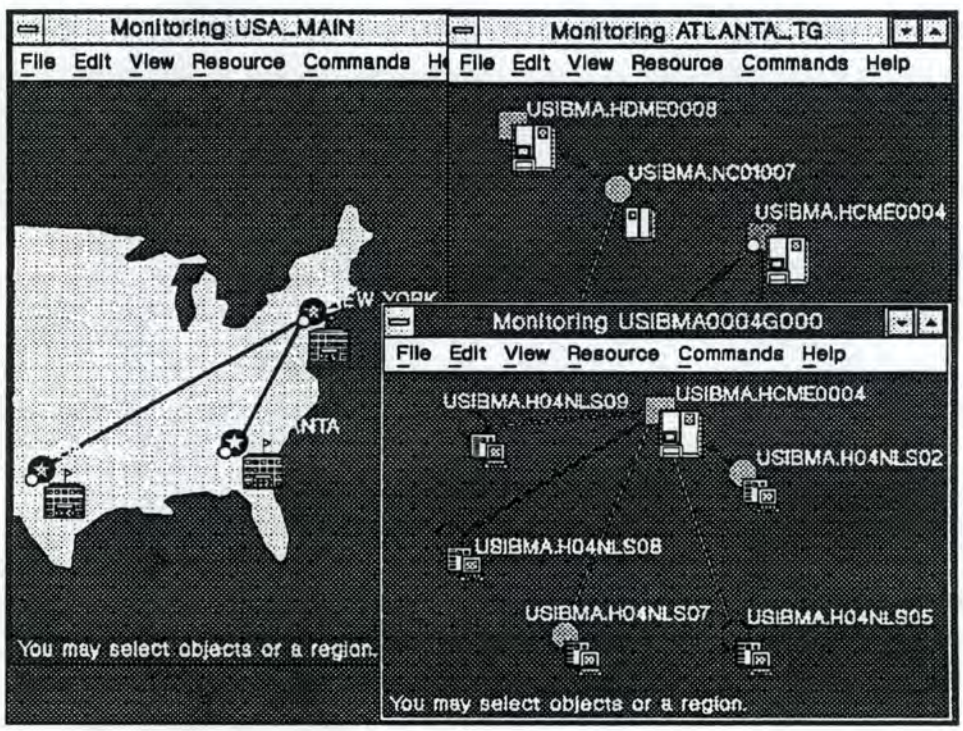


Figure 7.2. L'utilitaire "Graphic Monitor"

b) Le contrôleur "hardware"

Le contrôleur "hardware" récolte et enregistre de l'information à propos des ressources du réseau en panne. Cette information comprend des statistiques, des événements et des alertes.
De plus, le contrôleur "hardware" analyse les informations qu'il reçoit et présente à l'opérateur les causes probables de la panne ainsi que les actions à entreprendre recommandées.
Nous pourrions sans doute trouver un point commun entre les fonctionnalités du contrôleur "hardware" et celles des "Statistic Manager" et "Event Manager" du logiciel SPECTRUM.

c) Le contrôleur de statut

Le contrôleur de statut est un outil qui, en recevant les messages "système", permet d'identifier un composant en panne ou une erreur logique dans un programme. Il reçoit de l'information à propos du statut des ressources SNA figurant dans le réseau.
A côté de cela, le contrôleur de statut envoie l'information qu'il détient sur le statut des ressources à l'application "Graphic Monitor".

d) Le contrôleur de session

Le contrôleur de session récolte et enregistre de l'information complémentaire sur les sessions SNA entre deux ressources du réseau. Voici quelques exemples d'informations recueillies par le contrôleur de session: liaison coupée, erreur de protocole, disponibilité et configuration d'une session...

e) Le "Command Facility"

Le "Command Facility" constitue les opérations de base pour NETVIEW. Il exécute les commandes ainsi que les applications écrites par l'utilisateur. Le "Command Facility" peut également se charger d'exécuter automatiquement des commandes en réponse à la survenance de certains événements ou de certaines alertes.

Chacun de ces outils présente initialement de l'information à un haut niveau telle qu'une simple liste de toutes les alertes qui ont été détectées sur le réseau. En général, l'opérateur est capable de demander de plus en plus de détails sur un problème jusqu'à ce qu'il soit en mesure de l'isoler. Seule l'application "Graphic Monitor" propose une approche alternative pour isoler un problème en donnant à l'utilisateur l'option de se rendre directement au coeur du problème.

3. Conclusion

Dans ce chapitre, nous nous sommes contentés de présenter d'une manière générale deux logiciels de gestion de réseaux dominant actuellement le marché. Mais cette présentation reste purement théorique puisque nous n'avons pas eu l'occasion de tester aucun de ces deux logiciels. Les informations contenues dans ce chapitre ne proviennent donc que de documents écrits que nous nous sommes procurés auprès des sociétés Hewlett-Packard et I.B.M. situées toutes deux à Bruxelles.

Cependant, nous voudrions ajouter que les six mois passés au CERN sur le logiciel SPECTRUM nous ont fortement aidés à comprendre les grandes fonctionnalités des logiciels OPENVIEW et NETVIEW. En effet, les principales fonctionnalités d'un bon système de gestion de réseaux se retrouvent toutes dans ces trois logiciels, même si elles ne sont pas tout-à-fait identiques et ne possèdent pas la même appellation.

Nous aimerions, pour terminer, émettre une petite remarque sur les quelques comparaisons réalisées entre le logiciel SPECTRUM et les deux autres logiciels.
Ainsi, tout au long de ce chapitre, nous avons tenté de réaliser des comparaisons entre les fonctionnalités et les outils de SPECTRUM et ceux de OPENVIEW et NETVIEW. Mais comme nous l'avons déjà dit, nous n'avons pu tester aucun des deux logiciels; il se peut, par conséquent, que nous n'ayons pas tout-à-fait bien compris l'une ou l'autre de leurs fonctionnalités ou de leurs outils et que dès lors nos comparaisons avec SPECTRUM pourraient ne pas être totalement exactes.

CONCLUSION

Dans ce mémoire, nous avons d'abord présenté en détail le logiciel SPECTRUM avec ses grandes caractéristiques et ses principales fonctionnalités, ainsi que l'environnement du CERN dans lequel il sera amené à tourner. Ensuite, nous avons expliqué en long et en large les trois projets que nous avons dû développer durant notre stage. Nous avons également présenté quelques inconvénients du logiciel SPECTRUM qui nous ont posé des problèmes que nous avons dû contourner lors de la réalisation des projets.
Et finalement, nous avons donné une description succincte de deux autres logiciels de gestion de réseaux existant actuellement sur le marché en essayant parfois de réaliser des comparaisons avec SPECTRUM.

En guise de conclusion générale, nous pouvons dire que les six mois que nous avons passés au CERN nous ont plongés d'une manière pratique au coeur même des réseaux informatiques et des problèmes liés à leur gestion.

Durant toute la durée de notre stage, nous avons pu constater que les logiciels, les équipements, les composants et les supports utilisés pour la "communication d'information" par réseaux étaient loin d'être technologiquement parfaits. Le risque de voir apparaître un problème au niveau d'un composant du réseau reste encore relativement élevé.

Cela nous a permis de prendre conscience de l'importance de disposer d'un logiciel de gestion de réseaux informatiques performant qui permette de détecter au plus vite tous les problèmes au niveau d'un réseau, d'y remédier et ainsi de satisfaire des utilisateurs de plus en plus nombreux et de plus en plus exigeants.

BIBLIOGRAPHIE

[BOBA95] Boulanger, B., Bastin, P.,
"*Présentation du protocole de gestion de réseaux SNMPv2, Comparaison à SNMPv1*",
travail réalisé dans le cadre du cours de "Téléinformatique, Matière approfondie" donné par le prof. Ph. van Bastelaer de l'Institut d'Informatique des FUNDP,
1995, Namur (Belgique).

[CERN91] CERN, Commission Européenne pour la Recherche Nucléaire,
"*Le CERN, un exemple de coopération scientifique internationale.*",
avril 1991, CERN (Genève, Suisse)

[GAM88] Gamble, J.N.,
"*The Alarm System Display*",
October 1988, CERN (Genève, Suisse).

[GAM89] Gamble, J.N.,
"*Network Monitoring Reference Manual*",
September 1989, CERN (Genève, Suisse).

[GAM94] Gamble, J.N., Davids, D.,
"*Requirements for a Network Management System*",
19 May 1994, CERN (Genève, Suisse).

[HEY91] Heyvaert, D.,
"*Network Management Systems*",
31 janvier 1991, CERN (Genève, Suisse).

[HP1] Hewlett-Packard Company,
"*HP Network Connectivity Brochure.*",
1995, Netherlands.

[HP2] Hewlett-Packard Company,
"*HP J2561A OpenView Traffic Monitor/Windows HP J2562A OpenView Traffic Monitor/HP-UX.*",
November 1994, Netherlands.

[IBM1] I.B.M. Belgium s.a.,
"*IBM OS/2 - le super client*",
1995, Bruxelles, Belgique.

[IBM92] I.B.M. Corporation,
"*NetView: At a Glance.*",
May 1992, North Carolina (U.S.A.)

[INT94] William B., Norton (Merit)
Document Internet "*Cabletron SPECTRUM Pros/Cons*"
May 1994, USA.
Adresse électronique :
http://smurfland.cit.buffalo.edu/NetMan/ProdNotes/SpectrumNotes-WNorton.html

[NAC95] Nachtergaele, V.,
"*Description d'un protocole de gestion de réseau: le protocole SNMP.*",
Chapitre relatif à SNMP faisant partie du syllabus de "Téléinformatique, matière approfondie" de 2ème License et Maîtrise en Informatique,
1994, Namur (FUNDP), Belgique.

[NET93] Digital Equipment Corporation,
"*Polycenter Manager On Netview For Alpha AXP OSF/1 Users Guide Version V2.1 Field Test Draft*",
December 6, 1993, USA.

[ROSE91] Rose, M.,
"*The Simple Book: An Introduction to Management of TCP/IP-based internets*",
Prentice Hall Series in Innovative Technology,
1991, USA,
ISBN 0-13-812611-9.

[SPEC1] Cabletron Systems, Inc.,
"*SPECTRUM: Command Line Interface User's Guide*",
1993, USA.

[SPEC2] Cabletron Systems, Inc.,
"*SPECTRUM: Management Reports Guide*",
1993, USA.

[SPEC3] Cabletron Systems, Inc.,
"*Welcome to SPECTRUM!*",
1993, USA.

[SPEC4] Cabletron Systems, Inc.,
"*SPECTRUM: The EPI Module*",
1993, USA.

[SPEC5] Cabletron Systems, Inc.,
"*SPECTRUM: System User's Guide*",
1993, USA.

[SPEC6] Cabletron Systems, Inc.,
"*SPECTRUM: Administrator's Guide*",
1993, USA.

RFC utilisée :

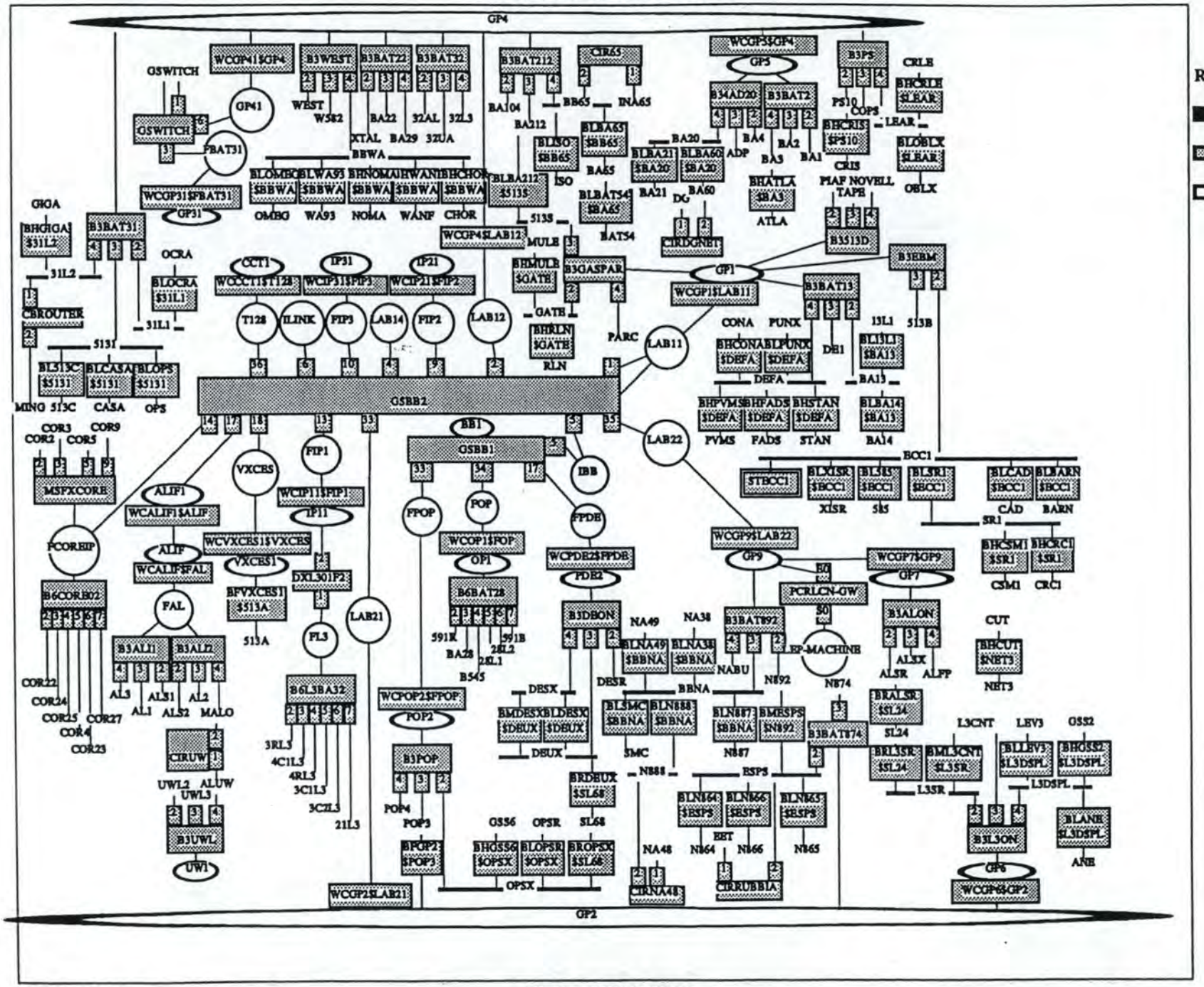
RFC 1213 : "*Management Information Base for Network Management of TCP/IP-based Internets: MIB-II.*"

ANNEXES:

ANNEXE 1 :

Topologie du réseau du CERN.

CERN INFRASTRUCTURE

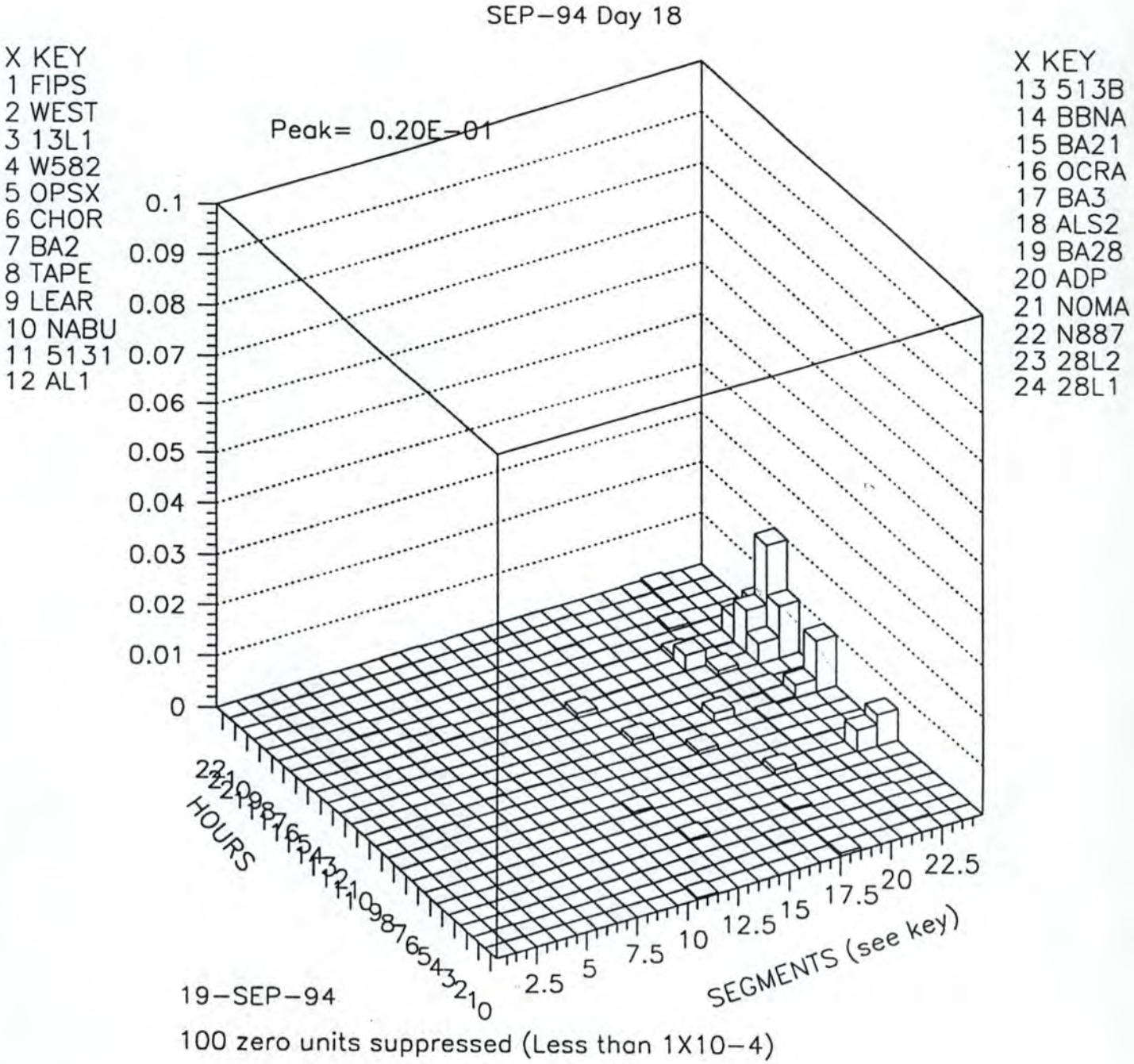


CAW CERN-CN/CS

Printed on 4-JAN-95 at 12:09

ANNEXE 2 :

Exemple de graphique fourni par l'application "Lego-Plot" sur les performances des composants du réseau.



100 zero units suppressed (Less than 1X10–4)

ANNEXE 3 :

Listing du programme "LectureDonnees".

```
#include <string.h>
#include <stdio.h>
#define NbreAttributs 6
#define Max_Ent1 1000
#define Max_Ent2 1000
#define Max_Ent3 5000


typedef struct table
 {
  int ident;
  char name[20];
 };

typedef struct moyenne_table
 {
  int ident;
  char hexa[9];
  char name[20];
 };


typedef struct grande_table
 {
  char dat[9];
  char heur[9];
  int identifiant;
  int attribut1;
  int attribut2;
  int attribut3;
  int attribut4;
  int attribut5;
  int attribut6;
 };


struct table tbl_Con[Max_Ent1];
struct moyenne_table tbl_Gen[Max_Ent2];
struct grande_table tbl_Stat[Max_Ent3];


/* PROCEDURE QUI TRANSFORME UN HEXADECIMAL EN DECIMAL     */


atoX (ptr)

register char *ptr;
{
 int value = 0;
 while (isxdigit (*ptr))
  {
   value = 16*value + (*ptr<='9' ? *ptr-'0': (*ptr<='F' ? *ptr+10-'A':
                       *ptr+10-'a'));
   ++ptr;
  }
 return (value);
}


main (LectureDonnees)

{
 typedef enum BOOL (false,true) bool;


 int i,j,k,l,m,n,numero_interface,tranche_horaire,int_h,nbr_Gen_IF_Port,compt,
     indice_Gen_IF_Port,int_hh,jj,nombre_elements,int_AN_Attr5[2], int_AN_Attr6[2],
     multi_broad_recus, multi_broad_emis, total_octets,pkt_rate,error_rate,error_in_rate
```



```
;


 char modele[18],interf[11],hh[3],ligne[512],id_nom[9],nom[40],command[512],
      hheure[9],hjour[3],hannee[3],hmois[4],identif[9],entier[8],maximum[12],
      ch[512],ch1[512],ch2[512],ch3[512],ch4[512],hexadecimal[7],

      date[9],heure[9],
      heure_error_rate[9],heure_total_octets[9],heure_pkt_rate[9],
      heure_error_in_rate[9],heure_multi_broad_emis[9],heure_multi_broad_recus[9],
      h1[6],h2[6],h3[6],h4[6],h5[6],h6[6],h7[6],h8[6],separateur[25],ih[3],
      attr1[12],attr2[12],attr3[12],attr4[12],attr5[12],attr6[12];


 float col1,col2,col3,col4,col5,col6,col7,col8,Delta_intAttr5,Delta_intAttr6;


 char *x,*p = hexadecimal, *q = id_nom;

 bool trouve,fini,Fichier_OK,premiere,premiere_fois,trop_grand;

 FILE *d,*h,*f,*fopen();


system ("/home/suncoms/spectrum/vnmsh/connect");



/* MISE EN MEMOIRE CENTRALE DES INTERFACES DANS LA TABLE 'tbl_Gen'          */


system ("/home/suncoms/spectrum/vnmsh/show models | grep Gen_IF_Port > resultat");

f = fopen ("resultat","r");
if (f == NULL) printf ("Erreur d'ouverture du fichier 'resultat'.");

nbr_Gen_IF_Port = 0;
trop_grand = false;
while (fgets(ligne,512,f) != NULL)
 {
  if (nbr_Gen_IF_Port <= Max_Ent2 - 1)
   {
    for (i=2;i<=7;i++) id_nom[i-2] = ligne[i];
    id_nom[6] = '\0';
    i = 9;
    strcpy (tbl_Gen[nbr_Gen_IF_Port].hexa,id_nom);
    tbl_Gen[nbr_Gen_IF_Port].ident = atoX (q);
    while ((ligne[i] != ' ') || (ligne[i+1] != ' '))
     {
      nom[i-9] = ligne[i];
      i++;
     }
    nom[i-9] = '\0';
    strcpy (tbl_Gen[nbr_Gen_IF_Port].name,nom);
   }
  else trop_grand = true;
  nbr_Gen_IF_Port = nbr_Gen_IF_Port + 1;
 }
if (trop_grand == true)
   {
    printf ("\nLa longueur de la table des interfaces 'tbl_Gen' est trop petite. \n");
    printf ("Veuillez modifier la valeur de la constante 'Max_Ent2'. Nouvelle valeur = %
d.\n",nbr_Gen_IF_Port);
   }
else printf ("\nTABLE 'tbl_Gen' CREEE.\n");
fclose (f);
```

```
/* MISE EN MEMOIRE CENTRALE DES CONNEXIONS ENTRE INTERFACES ET DEVICES     */
/* DANS LA TABLE 'tbl_Con'.                                                */


j = 0;
compt = 0;
trop_grand = false;
while (j <= nbr_Gen_IF_Port - 1)
 {
  printf ("\n%d. Device: %s ",j,tbl_Gen[j].name);
  sprintf (command,"/home/suncoms/spectrum/vnmsh/show children rel=Connects_to mh=%s > r
esultat",tbl_Gen[j].hexa);
  system (command);

  f = fopen ("resultat","r");
  if (f == NULL) printf ("Erreur d'ouverture du fichier 'resultat'.");

  fgets(ligne,512,f);
  while (fgets(ligne,512,f) != NULL)
   {
    if (compt <= Max_Ent1 - 1)
     {
      for (i=0;i<=7;i++) id_nom[i] = ligne[i];
      id_nom[8] = '\0';
      i = 9;
      tbl_Con[compt].ident = atoX (q);
      while ((ligne[i] != ' ') || (ligne[i+1] != ' '))
       {
        nom[i-9] = ligne[i];
        i++;
       }
      nom[i-9] = '\0';
      strcpy(tbl_Con[compt].name,nom);
      printf ("est relie a %s.",nom);
     }
    else trop_grand = true;
    compt++;
   }
  j++;
  fclose (f);
 }
fclose (d);
if (trop_grand == true)
   {
    printf ("\nLa longueur de la table des connexions 'tbl_Con' est trop petite. \n");
    printf ("Veuillez modifier la valeur de la constante 'Max_Ent1'. Nouvelle valeur = %
d.\n",compt);
   }
else printf ("\n\nTABLE 'tbl_Con' CREEE.\n\n");



system ("/home/suncoms/spectrum/vnmsh/disconnect");


/* LECTURE DE LA 4e LIGNE AVANT LA FIN DU FICHIER 'exportdata.log' POUR */
/* OBTENIR L'HEURE ET LA DATE DE L'EXPORTATION.                         */


d = fopen ("exportdata.log","r");
if (d == NULL) printf ("Erreur fichier exportdata.log");
i = 1;
while (fgets(ch,512,d) != NULL)
 {
  if (i == 1)
   {
    strcpy (ch1,ch);
```



```
    j = 1;
   }
  if (i == 2)
   {
    strcpy (ch2,ch);
    j = 2;
   }
  if (i == 3)
   {
    strcpy (ch3,ch);
    j = 3;
   }
  if (i == 4)
   {
    strcpy (ch4,ch);
    j = 4;
    i = 0;
   }
  i++;
 }
if (j == 1) strcpy (ligne,ch2);
  else if (j == 2) strcpy (ligne,ch3);
    else if (j == 3) strcpy (ligne,ch4);
      else strcpy (ligne,ch1);
fclose(d);
for (i=34;i<37;i++) hmois[i-34] = toupper(ligne[i]);
hmois[3] = '\0';
for (i=38;i<40;i++)
 {
  if (ligne[i] == ' ') hjour[i-38] = '0';
    else hjour[i-38] = ligne[i];
 }
hjour[2] = '\0';
for (i=52;i<54;i++) hannee[i-52] = ligne[i];
hannee[2] = '\0';



/* DECHARGEMENT EN MEMOIRE CENTRALE DE 'stat_Day' PAR TRANCHE D'HEURE */


d = fopen ("stat_Day","r");
if (d == NULL) printf ("Erreur fichier stat_Day");
sprintf (maximum,"TABLE.LOG%s",hjour);
h = fopen (maximum,"w");
if (h == NULL) printf ("Erreur fichier maximum.logNN");
Fichier_OK = true;
premiere_fois = true;
premiere = true;
while (Fichier_OK == true)
{
 jj = 0;
 fini = false;
 if (premiere_fois == true)
  {
   x = fgets (ligne,512,d);
   premiere_fois = false;
  }
 while ((x != NULL) && (fini == false))
 {
  if (jj > Max_Ent3 - 1) printf ("Erreur fichier 'stat_Day' trop long.\n");
  for (i=9;i<11;i++) hh[i-9] = ligne[i];
  hh[2] = '\0';
  int_hh = atoi (hh);
  if (jj == 0)
   {
    tranche_horaire = int_hh;
    for (i=9;i<14;i++) {h1[i-9] = ligne[i]; h8[i-9] = ligne[i];}
    h1[5] = '\0'; h8[5] = '\0';
```

```
    for (i=12;i<17;i++) hheure[i-12] = ligne[i];
    hheure[5] = '\0';
    for (i=9;i<11;i++) ih[i-9] = ligne[i];
    ih[2] = '\0';
    int_h = atoi(ih);
    int_h++;
    if (premiere == false)
     {
      strcpy (separateur,"----------------------");
      fprintf (h,"%s\n",separateur);
     }
    else premiere = false;
    fprintf (h,"Information dumped at %s-%s-%s %02d:%s\n",hjour,hmois,hannee,int_h,hheur
e);
   }
  if (int_hh == tranche_horaire)
   {
    for (i=0;i<8;i++) tbl_Stat[jj].dat[i] = ligne[i];
    tbl_Stat[jj].dat[8] = '\0';
    for (i=9;i<17;i++) tbl_Stat[jj].heur[i-9] = ligne[i];
    tbl_Stat[jj].heur[8] = '\0';
    if (jj == 0) strcpy (hheure,tbl_Stat[jj].heur);
    for (i=20;i<26;i++) hexadecimal[i-20] = ligne[i];
    hexadecimal[6] = '\0';
    tbl_Stat[jj].identifiant = atoX (p);  ;
    i = 27;
    while (ligne[i] != ',')
     {
      attr1[i-27] = ligne[i];
      i++;
     }
     attr1[i-27] = '\0';
     tbl_Stat[jj].attribut1 = (float)(atoi(attr1));
    i++;  /* POUR PASSER LA VIRGULE QUI SEPARE LES ATTRIBUTS          */
    k = 0;
    while (ligne[i] != ',')
     {
      attr2[k] = ligne[i];
      i++;
      k++;
     }
    attr2[k] = '\0';
    tbl_Stat[jj].attribut2 = (float)(atoi(attr2));
    i++;  /* POUR PASSE LA VIRGULE QUI SEPARE LES ATTRIBUTS           */
    k = 0;
    while (ligne[i] != ',')
     {
      attr3[k] = ligne[i];
      i++;
      k++;
     }
    attr3[k] = '\0';
    tbl_Stat[jj].attribut3 = (float) (atoi(attr3));
    i++;  /* POUR PASSER LA VIRGULE QUI SEPARE LES ATTRIBUTS          */
    k = 0;
    while (ligne[i] != ',')
     {
      attr4[k] = ligne[i];
      i++;
      k++;
     }
    attr4[k] = '\0';
    tbl_Stat[jj].attribut4 = (float) (atoi(attr4));
    i++;  /* POUR PASSER LA VIRGULE QUI SEPARE LES ATTRIBUTS          */
    k = 0;
    while (ligne[i] != ',')
     {
      attr5[k] = ligne[i];
      i++;
```



```
      k++;
     }
    attr5[k] = '\0';
    tbl_Stat[jj].attribut5 = (float) (atoi(attr5));
    i++;  /* POUR PASSER LA VIRGULE QUI SEPARE LES ATTRIBUTS          */
    k = 0;
    while (ligne[i] != '\n')
     {
      attr6[k] = ligne[i];
      i++;
      k++;
     }
    attr6[k] = '\0';
    tbl_Stat[jj].attribut6 = (float) (atoi(attr6));
    jj++;
    x = fgets (ligne,512,d);
    if (x == NULL) Fichier_OK = false;
   }
  else
   {
    fini = true;
    tranche_horaire = int_hh;
    for (i=0;i<2;i++) hheure[i] = hh[i];
   }
}
nombre_elements = jj;



/* LECTURE DE 'tbl_Stat' POUR OBTENIR, POUR CHAQUE Gen_IF_Port,  */
/* SES MEILLEURES PERFORMANCES (AVEC DATE ET HEURE), ET ECRIT CELLES-CI  */
/* DANS LE FICHIER 'maximum.log' TRIEES SUR L'HEURE.                     */


indice_Gen_IF_Port = 0;
while (indice_Gen_IF_Port <= nbr_Gen_IF_Port - 1)
 {
  trouve = false;
  j = 0;
  while ((trouve == false) && (j <= Max_Ent1 - 1))
   {
    if (tbl_Gen[indice_Gen_IF_Port].ident == tbl_Con[j].ident)
     {
      trouve = true;
      strcpy (interf,tbl_Con[j].name);
     }
    j++;
   }
  if (trouve == false) strcpy (interf,"Unnamed");
  strcpy (modele,tbl_Gen[indice_Gen_IF_Port].name);
  error_in_rate = 0;
  error_rate = 0;
  total_octets = 0;
  multi_broad_emis = 0;
  multi_broad_recus = 0;
  pkt_rate = 0;
  numero_interface = 0;
  jj = 0;
  while (jj <= nombre_elements-1)
   {                              /* parcours de la table 'stat_Day' */
                 /* Verification que la ligne du fichier 'stat_Day'      */
                 /* correspond au numero d'une ligne du fichhier         */
                 /* 'Gen_IF_Port'                                        */

    if (tbl_Stat[jj].identifiant == tbl_Gen[indice_Gen_IF_Port].ident)
     {
      numero_interface++; /* (numero_interface)e ligne ds la fichier    */
                          /* 'stat_Day' correspondant au number_interface */
```

```
      if (numero_interface == 1)
       {
        int_AN_Attr5[0] = tbl_Stat[jj].attribut5;  /* X_InNUCast          */
        int_AN_Attr6[0] = tbl_Stat[jj].attribut6;  /* X_OutNUCast         */
        Delta_intAttr5 = 0;
        Delta_intAttr6 = 0;
       }
        else
          {
           if (numero_interface != 2)
            {
             int_AN_Attr5[0] = int_AN_Attr5[1];
             int_AN_Attr6[0] = int_AN_Attr6[1];
            }
           int_AN_Attr5[1] = tbl_Stat[jj].attribut5;
           int_AN_Attr6[1] = tbl_Stat[jj].attribut6;
           Delta_intAttr5 = (int_AN_Attr5[1] - int_AN_Attr5[0]);
           Delta_intAttr6 = (int_AN_Attr6[1] - int_AN_Attr6[0]);
          }
      if (tbl_Stat[jj].attribut1 >= total_octets)
       {
        total_octets = tbl_Stat[jj].attribut1;
        strcpy (heure_total_octets,tbl_Stat[jj].heur);
       }
      if (tbl_Stat[jj].attribut2 >= error_rate)
       {
        error_rate = tbl_Stat[jj].attribut2;
        strcpy (heure_error_rate,tbl_Stat[jj].heur);
       }
      if (tbl_Stat[jj].attribut3 >= error_in_rate)
       {
        error_in_rate = tbl_Stat[jj].attribut3;
        strcpy (heure_error_in_rate,tbl_Stat[jj].heur);
       }
      if (tbl_Stat[jj].attribut4 >= pkt_rate)
       {
        pkt_rate = tbl_Stat[jj].attribut4;
        strcpy (heure_pkt_rate,tbl_Stat[jj].heur);
       }
      if (Delta_intAttr5 >= multi_broad_recus)
       {
        multi_broad_recus = Delta_intAttr5;
        strcpy (heure_multi_broad_recus,tbl_Stat[jj].heur);
       }
      if (Delta_intAttr6 >= multi_broad_emis)
       {
        multi_broad_emis = Delta_intAttr6;
        strcpy (heure_multi_broad_emis,tbl_Stat[jj].heur);
       }
     }
    jj++;
   }
  if (numero_interface != 0)
   {
    col1 = 0;
    col2 = pkt_rate; strncpy (h2,heure_pkt_rate,5); h2[5] = '\0';
    col3 = error_in_rate; strncpy (h3,heure_error_in_rate,5); h3[5] = '\0';
    col4 = error_rate; strncpy (h4,heure_error_rate,5); h4[5] = '\0';
    col5 = total_octets; strncpy (h5,heure_total_octets,5); h5[5] = '\0';
    col6 = multi_broad_recus; strncpy (h6,heure_multi_broad_recus,5); h6[5] = '\0';
    col7 = multi_broad_emis; strncpy (h7,heure_multi_broad_emis,5); h7[5] = '\0';
    col8 = 0;
    fprintf (h,"%-19s %-10s %4.3E %5s  %4.3E %5s  %4.3E %5s  %4.3E %5s  %4.3E %5s  %4.3E
 %5s  %4.3E %5s  %4.3E %5s\n",
            modele,interf,col1,h1,col2,h2,col3,h3,col4,h4,col5,h5,col6,h6,col7,h7,col8,
h8);
    }
   indice_Gen_IF_Port++;
   }
```



```
  }
fclose (h);
fclose (d);
}
```

ANNEXE 4 :

Listing des programmes constituant le projet "AlarmMirror" :

- **Set_Alarm**
- **Update_Alarm**
- **Cancel_Alarm**

```
/* PROGRAMME LANCE LORS DE CHAQUE NOUVELLE ALARME                          */
/* 'WA_SEGMENT' DE COULEUR ROUGE -> id = AL.                               */
/*                        ORANGE -> id = WA.                               */
/* 'Coax_SEGMENT' DE COULEUR ROUGE -> id = AL.                             */
/*                          ORANGE -> id = WA.                             */
/* POUR TOUS LES AUTRES: DE COULEUR ROUGE OU ORANGE -> id = AL.            */
/*                       DE COULEUR JAUNE OU GRISE -> id = WA.             */
/* IL L'ENVOIE A 'ALARM_MIRROR' SUR LE PORT DE SOCKET #3210 SUR Uxcsb1.    */


#include <stdio.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <netdb.h>
#include <sys/socket.h>



/* PROCEDURE QUI TRANSFORME UN HEXADECIMAL EN DECIMAL      */

atoX (ptr)

register char *ptr;
{
 int value = 0;
 while (isxdigit (*ptr))
  {
   value = 16*value + (*ptr<='9' ? *ptr-'0': (*ptr<='F' ? *ptr+10-'A':
                       *ptr+10-'a'));
   ++ptr;
  }
 return (value);
}



main (argc,argv)

 int argc;
 char *argv[];

{
typedef enum BOOL (true,false) bool;

char data[500],id[4],system[10],type[15],modele[20],alarm_id[8],code_cause[12],
     numero[10],ligne[1024],explication[60],coul[10],coul1[10],coul2[10],
     coul3[10],coul4[10],WA[15],Fanout[15],Coax[15],gns[15],Rtr[20],Gen[20],
     TCP1[20],IP1[20],IP2[20],UDP1[20],UDP2[20],Static[20],System1[20],ICMP[20],
     Shiva[20],GenRtr[20],SNMP1[20],SNMP2[20],Trans[20],Apple[20],System2[20];

char *ptr;

int i,j,l,m,sock,status,int_code_cause,k,int_num;

bool trouve;

struct sockaddr_in sin;

FILE *f,*fopen();


strcpy (code_cause,argv[4]);
ptr = code_cause;
int_code_cause = atoX (ptr);
strcpy (UDP1,"UDP1_App");
strcpy (UDP2,"UDP2_App");
strcpy (System1,"System1_App");
strcpy (System2,"System2_App");
```



```
strcpy (ICMP,"ICMP_App");
strcpy (Trans,"Transparnt_App");
strcpy (TCP1,"TCP1_App");
strcpy (Apple,"ApplTlkRtrApp");
strcpy (Static,"Static_App");
strcpy (IP1,"IP1_App");
strcpy (IP2,"IP2_App");
strcpy (SNMP1,"SNMP1_Agent");
strcpy (SNMP2,"SNMP2_Agent");
strcpy (GenRtr,"GenRtrApp");
strcpy (Shiva,"Shiva_Dot3_App");
strcpy (Gen,"Gen_IF_Port");
strcpy (Rtr,"Rtr_Shiva_FP5");

if ((argc == 6) && (strcmp (argv[1],UDP1) != 0) && (strcmp (argv[1],System1) != 0) && (s
trcmp (argv[1],ICMP) != 0)
   && (strcmp (argv[1],Static) != 0) && (strcmp (argv[1],IP1) != 0) && (strcmp (argv[1],
IP2) != 0) && (strcmp (argv[1],UDP2) != 0)
   && (strcmp (argv[1],SNMP1) != 0) && (strcmp (argv[1],Trans) != 0) && (strcmp (argv[1]
,SNMP2) != 0) && (strcmp (argv[1],GenRtr) != 0)
   && (strcmp (argv[1],System2) != 0) && (strcmp (argv[1],Apple) != 0) && (strcmp (argv[
1],Shiva) != 0)
   && (strcmp (argv[1],Rtr) != 0) && (strcmp (argv[1],Gen) != 0) && (strcmp (argv[1],TCP
1) != 0))
{
strcpy (coul1,"RED");
strcpy (coul2,"ORANGE");
strcpy (coul3,"YELLOW");
strcpy (coul4,"GRAY");
strcpy (Fanout,"Fanout");
strcpy (WA,"WA_Segment");
                                /* ON S'OCCUPE DES 'Coax_Segment' ET DES */
strcpy (Coax,"Coax_Segment"); /* 'WA_Segment' DONT LA CAUSE N'EST PAS "Contact_Lost" */
trouve = true;
strcpy (coul,argv[5]);
strcpy (type,argv[1]);

if ( ( ( ((strcmp(type,Coax) == 0) || (strcmp(type,WA) == 0)) && (int_code_cause != 6554
5) )

                                        ||

                                (strcmp(type,Fanout) == 0) )

                                        &&

                                (strcmp(coul,coul1) == 0) )
                                   strcpy (id,"AL:");

else if ( ( ( ((strcmp(type,Coax) == 0) || (strcmp(type,WA) == 0)) && (int_code_cause !=
 65545) )

                                        ||

                                (strcmp(type,Fanout) == 0) )

                                        &&

                                (strcmp(coul,coul2) == 0) )
                                  strcpy (id,"WA:");


  else if (((strcmp(coul,coul1) == 0) || (strcmp(coul,coul2) == 0)) && ((strcmp(type,Coa
x) != 0) && (strcmp(type,WA) != 0) && (strcmp(type,Fanout) != 0)))
              strcpy (id,"AL:");
    else if (((strcmp(coul,coul3) == 0) || (strcmp(coul,coul4) == 0)) && ((strcmp(type,C
oax) != 0) && (strcmp(type,WA) != 0) && (strcmp(type,Fanout) != 0)))
          strcpy (id,"WA:");
      else trouve = false;
```

```
if (trouve == true)
 {
  for (i=0;i<3;i++) data[i] = id[i];
  strcpy (system,"SPECMN:");
  for (i=0;i<7;i++) data[i+3] = system[i];
  i = 10;
  strcpy (alarm_id,argv[3]);
  j = 0;
  while (alarm_id[j] != '\0')
   {
    data[i] = alarm_id[j];
    j++;i++;
   }
  data[i] = ':';
  i++;
  j = 0;
  while (type[j] != '\0')
   {
    data[i] = type[j];
    j++;i++;
   }
  data[i] = ' ';
  i++;
  strcpy (modele,argv[2]);
  j = 0;
  while (modele[j] != '\0')
   {
    data[i] = modele[j];
    j++;i++;
   }
  data[i] = ' ';
  i++;
  strcpy (code_cause,argv[4]);
  ptr = code_cause;
  int_code_cause = atoX(ptr);


    trouve = false;
    f = fopen("Alarm_Code","r");
    if (f == NULL) printf ("Erreur fichier 'Alarm_Code'.\n");
    while ((fgets(ligne,1024,f)) && (trouve == false))
     {
      j = 0;
      while (ligne[j] != ' ')
       {
        numero[j] = ligne[j];
        j++;
       }
       numero[j] = '\0';
       int_num = atoi(numero);
       if (int_num == int_code_cause)
        {                              /* RECHERCHE DE L'EXPLICATION DE L'ALARME */
         j++;                          /* GRACE A SON CODE.                      */
         k = 0;                        /* (parcours du fichier 'Alarm_Code')     */
         trouve = true;
         while (ligne[j] != '\n')
          {
           explication[k] = ligne[j];
           k++;j++;
          }
         explication[k] = '\0';
        }
     }
    fclose(f);
    if (trouve == false) strcpy (explication,"UNKNOWN CAUSE...");
    j = 0;
    while (explication[j] != '\0')
     {
```



```
      data[i] = explication[j];
      j++;i++;
     }
    data[i] = '\0';

    sock = socket (AF_INET,SOCK_DGRAM,0);
    if (sock < 0)
     {
      perror ("Ouverture du socket");
      exit();
     }
    sin.sin_addr.s_net   = (unsigned char) 0x80;
    sin.sin_addr.s_host  = (unsigned char) 0x8d; /* Adresse IP Uxcsb1 */
    sin.sin_addr.s_lh    = (unsigned char) 0x0;
    sin.sin_addr.s_impno = (unsigned char) 0x2c;
    sin.sin_family = AF_INET;
    sin.sin_port = 3210;

    status = sendto (sock,data,sizeof(data),0,&sin,sizeof(sin));
    if (status < 0)
       perror ("envoi de la donnee");
    close (sock);
    printf ("%s\n",data);
   }
  }
}
```

```
/* PROGRAMME LANCE LORS DE CHAQUE MODIFICATION D'UNE ALARME EXISTANTE      */
/* DE COULEUR ROUGE (id = AL) OU ORANGE POUR UN 'COAX_SEGMENT' (id = WA),  */
/* DE COULEUR ROUGE ou ORANGE (id = AL), JAUNE OU GRISE (id = WA) POUR     */
/* TOUS LES AUTRES APPAREILS.                                              */
/* IL L'ENVOIE A 'ALARM-MIRROR' VIA LE SOCKET #3210 SUR Uxcsb1.            */



#include <stdio.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <netdb.h>
#include <sys/socket.h>



/* PROCEDURE QUI TRANSFORME UN HEXADECIMAL EN DECIMAL      */

atoX (ptr)

register char *ptr;
{
 int value = 0;
 while (isxdigit (*ptr))
  {
   value = 16*value + (*ptr<='9' ? *ptr-'0': (*ptr<='F' ? *ptr+10-'A':
                       *ptr+10-'a'));
   ++ptr;
  }
 return (value);
}



main (argc,argv)

 int argc;
 char *argv[];

{
typedef enum BOOL (true,false) bool;

char data[500],id[4],system[10],type[15],modele[20],alarm_id[8],code_cause[12],
     numero[10],ligne[1024],explication[60],coul1[10],coul2[10],coul3[10],
     coul[10],coul4[10],WA[15],Fan[15],Coax[15],gns[15],Gen[20],Rtr[20],
     TCP1[20],IP1[20],IP2[20],UDP1[20],UDP2[20],Static[20],System1[20],ICMP[20],
     System2[20],Apple[20],Shiva[20],GenRtr[20],SNMP1[20],SNMP2[20],Trans[20];

char *ptr;

int i,j,l,m,sock,status,int_code_cause,k,int_num;

bool trouve;

struct sockaddr_in sin;

FILE *f,*g,*h,*fopen();


strcpy (code_cause,argv[4]);
ptr = code_cause;
int_code_cause = atoX(ptr);
strcpy (UDP1,"UDP1_App");
strcpy (UDP2,"UDP2_App");
strcpy (System1,"System1_App");
strcpy (System2,"System2_App");
strcpy (TCP1,"TCP1_App");
strcpy (Apple,"AppleTlkRtrApp");
strcpy (Shiva,"Shiva_Dot3_App");
```



```
strcpy (ICMP,"ICMP_App");
strcpy (Trans,"Transparnt_App");
strcpy (Static,"Static_App");
strcpy (IP1,"IP1_App");
strcpy (IP2,"IP2_App");
strcpy (SNMP1,"SNMP1_Agent");
strcpy (SNMP2,"SNMP2_Agent");
strcpy (GenRtr,"GenRtrApp");
strcpy (Gen,"Gen_IF_Port");
strcpy (Rtr,"Rtr_Shiva_FP5");

if ((argc == 6) && (strcmp (argv[1],UDP1) != 0) && (strcmp (argv[1],System1) != 0) && (s
trcmp (argv[1],ICMP) != 0)
   && (strcmp (argv[1],Static) != 0) && (strcmp (argv[1],IP1) != 0) && (strcmp (argv[1],
IP2) != 0) && (strcmp (argv[1],UDP2) != 0)
   && (strcmp (argv[1],SNMP1) != 0) && (strcmp (argv[1],Trans) != 0) && (strcmp (argv[1]
,SNMP2) != 0) && (strcmp (argv[1],GenRtr) != 0)
   && (strcmp (argv[1],System2) != 0) && (strcmp (argv[1],Shiva) != 0) && (strcmp (argv[
1],Apple) != 0)
   && (strcmp (argv[1],Rtr) != 0) && (strcmp (argv[1],Gen) != 0) && (strcmp(argv[1],TCP1
) != 0))
{
strcpy (coul1,"RED");
strcpy (coul2,"ORANGE");
strcpy (coul3,"YELLOW");
strcpy (coul4,"GRAY");
strcpy (coul,argv[5]);
strcpy (type,argv[1]);
strcpy (WA,"WA_Segment");
strcpy (Fan,"Fanout");
strcpy (Coax,"Coax_Segment"); /* ON S'OCCUPE DES 'WA_Segment' ET DES   */
                              /* 'Coax_Segment' DONT LA CAUSE D'ALAME N'EST   */
trouve = true;                /* PAS UN 'Contact Lost'.                       */

if ( ( ( ((strcmp(type,Coax) == 0) || (strcmp(type,WA) == 0)) && (int_code_cause != 6554
5) ) || (strcmp(type,Fan) == 0) ) &&
   (strcmp(coul,coul1) == 0)) strcpy (id,"AL:");
else if ((((strcmp(type,Coax) == 0) || (strcmp(type,WA) == 0)) && (int_code_cause != 65
545)) || (strcmp(type,Fan) == 0)) &&
        (strcmp(coul,coul2) == 0)) strcpy (id,"WA:");
  else if (((strcmp(coul,coul1) == 0) || (strcmp(coul,coul2) == 0)) && ((strcmp(type,Coa
x) != 0) && (strcmp(type,WA) != 0) && (strcmp(type,Fan) != 0)))
         strcpy (id,"AL:");
    else if (((strcmp(coul,coul3) == 0) || (strcmp(coul,coul4) == 0)) && ((strcmp(type,C
oax) != 0) && (strcmp(type,WA) != 0) && (strcmp(type,Fan) != 0)))
           strcpy (id,"WA:");
      else trouve = false;

if (trouve == true)
 {
  for (i=0;i<3;i++) data[i] = id[i];
  strcpy (system,"SPECMN:");
  for (i=0;i<7;i++) data[i+3] = system[i];
  j = 0;
  i = 10;
  strcpy (alarm_id,argv[3]);
  j = 0;
  while (alarm_id[j] != '\0')
   {
    data[i] = alarm_id[j];
    j++;i++;
   }
  data[i] = ':';
  i++;
  j = 0;
  while (type[j] != '\0')
   {
    data[i] = type[j];
    j++;i++;
```

```
  )
 data[i] = ' ';
 i++;
 strcpy (modele,argv[2]);
 j = 0;
 while (modele[j] != '\0')
  {
   data[i] = modele[j];
   j++;i++;
  }
 data[i] = ' ';
 i++;
 trouve = false;
 f = fopen("Alarm_Code","r");
 if (f == NULL) printf ("Erreur fichier 'Alarm_Code'.\n");
 while ((fgets(ligne,1024,f)) && (trouve == false))
  {
   j = 0;
   while (ligne[j] != ' ')  /* RECHERCHE DE L'EXPLICATION DE L'ALARME    */
    {                       /* EN FONCTION DU CODE DE LA CAUSE.          */
     numero[j] = ligne[j];  /* (parcours du fichier 'Alarm_Code')        */
     j++;
    }
   numero[j] = '\0';
   int_num = atoi(numero);
   if (int_num == int_code_cause)
    {
     j++;
     k = 0;
     trouve = true;
     while (ligne[j] != '\n')
      {
       explication[k] = ligne[j];
       k++;j++;
      }
     explication[k] = '\0';
    }
  }
 fclose(f);
 if (trouve == false) strcpy (explication,"UNKNOWN CAUSE...");
 j = 0;
 while (explication[j] != '\0')
  {
   data[i] = explication[j];
   j++;i++;
  }
 data[i] = '\0';

 sock = socket (AF_INET,SOCK_DGRAM,0);
 if (sock < 0)
  {
   perror ("Ouverture du socket");
   exit();
  }
 sin.sin_addr.s_net   = (unsigned char) 0x80;
 sin.sin_addr.s_host  = (unsigned char) 0x8d;
 sin.sin_addr.s_lh    = (unsigned char) 0x0; /* Adresse IP de Uxcsb1 */
 sin.sin_addr.s_impno = (unsigned char) 0x2c;
 sin.sin_family = AF_INET;
 sin.sin_port = 3210;

 status = sendto (sock,data,sizeof(data),0,&sin,sizeof(sin));
 if (status < 0)
    perror ("envoi de la donnee");
 close (sock);
 printf ("%s\n",data);
)
}
}
```

```
/* PROGRAMME LANCE LORS DE CHAQUE ANNULATION D'UNE ALARME EXISTANTE (id = CA)*/
/* DE COULEUR ROUGE OU ORANGE POUR UN 'COAX_SEGMENT',                        */
/* DE COULEUR ROUGE, ORANGE, JAUNE OU GRISE POUR UN POUR TOUS LES AUTRES     */
/* APPAREILS.                                                                */
/* IL L'ENVOIE A 'ALARM-MIRROR' VIA LE SOCKET #3210 SUR Uxcsb1.              */


#include <stdio.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <netdb.h>
#include <sys/socket.h>


main (argc,argv)

 int argc;
 char *argv[];

{
typedef enum BOOL {true,false} bool;

char data[500],id[4],system[10],type[15],modele[20],alarm_id[8],code_cause[12],
     numero[10],ligne[1024],
     coul1[10],coul2[10],coul3[10],coul4[10],WA[15],Fan[15],Coax[15],gns[15],
     coul[10],contact_lost[10],Gen[20],Rtr[20],TCP1[20],
     IP1[20],IP2[20],UDP1[20],UDP2[20],Static[20],System1[20],ICMP[20],
     Apple[20],GenRtr[20],SNMP1[20],SNMP2[20],Trans[20],System2[20],Shiva[20];

int i,j,sock,status;

bool trouve;

struct sockaddr_in sin;


strcpy (contact_lost,"10009");
strcpy (UDP1,"UDP1_App");
strcpy (UDP2,"UDP2_App");
strcpy (System1,"System1_App");
strcpy (System2,"System2_App");
strcpy (Apple,"ApplTlkRtrApp");
strcpy (ICMP,"ICMP_App");
strcpy (Trans,"Transparnt_App");
strcpy (Static,"Static_App");
strcpy (IP1,"IP1_App");
strcpy (IP2,"IP2_App");
strcpy (TCP1,"TCP1_App");
strcpy (SNMP1,"SNMP1_Agent");
strcpy (SNMP2,"SNMP2_Agent");
strcpy (GenRtr,"GenRtrApp");
strcpy (Shiva,"Shiva_Dot3_App");
strcpy (Gen,"Gen_IF_Port");
strcpy (Rtr,"Rtr_Shiva_FP5");

if ((argc == 6) && (strcmp (argv[1],UDP1) != 0) && (strcmp (argv[1],System1) != 0) && (s
trcmp (argv[1],ICMP) != 0) && (strcmp (argv[1],System2) != 0)
   && (strcmp (argv[1],Static) != 0) && (strcmp (argv[1],IP1) != 0) && (strcmp (argv[1],
IP2) != 0) && (strcmp (argv[1],UDP2) != 0)
   && (strcmp (argv[1],SNMP1) != 0) && (strcmp (argv[1],Trans) != 0) && (strcmp (argv[1]
,SNMP2) != 0) && (strcmp (argv[1],GenRtr) != 0)
   && (strcmp (argv[1],Apple) != 0) && (strcmp (argv[1],Shiva) != 0) && (strcmp (argv[1]
,Rtr) != 0) && (strcmp (argv[1],Gen) != 0)
   && (strcmp (argv[1],TCP1) != 0))
{
strcpy (coul1,"RED");
strcpy (coul2,"ORANGE");
```



```
strcpy (coul3,"YELLOW");
strcpy (coul4,"GRAY");
strcpy (WA,"WA_Segment");
strcpy (Fan,"Fanout");
strcpy (Coax,"Coax_Segment"); /* ON S'OCCUPE DES 'WA_Segment' et des       */
                              /* 'Coax_Segment' DONT LA CAUSE DE L'ALARME  */
strcpy (type,argv[1]);        /* EST DIFFERENTE DE 'Contact_Lost'          */
strcpy (coul,argv[5]);
strcpy (code_cause,argv[4]);

if (((((strcmp(type,Coax) == 0) || (strcmp(type,WA) == 0)) && (strcmp (contact_lost,argv
[4]) != 0)) || (strcmp(type,Fan) == 0)) &&
   (strcmp(coul,coul1) == 0)) trouve = true;
  else if ((((((strcmp(type,Coax) == 0) || (strcmp(type,WA) == 0)) && (strcmp (contact_lo
st,argv[4]) != 0)) || (strcmp(type,Fan) == 0)) &&
        (strcmp(coul,coul2) == 0)) trouve = true;
    else if ((((strcmp(coul,coul1) == 0) || (strcmp(coul,coul2) == 0))) && ((strcmp(type
,Coax) != 0) && (strcmp(type,WA) != 0) && (strcmp(type,Fan) != 0)))
            trouve = true;
      else if ((((strcmp(coul,coul3) == 0) || (strcmp(coul,coul4) == 0))) && ((strcmp(ty
pe,Coax) != 0) && (strcmp(type,WA) != 0) && (strcmp(type,Fan) != 0)))
              trouve = true;
        else trouve = false;


if (trouve == true)
{
 strcpy (id,"CA:");
 for (i=0;i<3;i++) data[i] = id[i];
 strcpy (system,"SPECMN:");
 for (i=0;i<7;i++) data[i+3] = system[i];
 j = 0;
 i = 10;
 strcpy (alarm_id,argv[3]);
 j = 0;
 while (alarm_id[j] != '\0')
  {
   data[i] = alarm_id[j];
   j++;i++;
  }
 data[i] = ':';
 i++;
 j = 0;
 while (type[j] != '\0')
  {
   data[i] = type[j];
   j++;i++;
  }
 data[i] = ' ';
 i++;
 strcpy (modele,argv[2]);
 j = 0;
 while (modele[j] != '\0')
  {
   data[i] = modele[j];
   j++;i++;
  }
 data[i] = '\0';

 sock = socket (AF_INET,SOCK_DGRAM,0);
 if (sock < 0)
  {
   perror ("Ouverture du socket");
   exit();
  }
 sin.sin_addr.s_net  = (unsigned char) 0x80;
 sin.sin_addr.s_host = (unsigned char) 0x8d;
 sin.sin_addr.s_lh   = (unsigned char) 0x0;  /* Adresse IP de Uxcsb1 */
 sin.sin_addr.s_impno = (unsigned char) 0x2c;
```

```
  sin.sin_family = AF_INET;
  sin.sin_port = 3210;

  status = sendto (sock,data,sizeof(data),0,&sin,sizeof(sin));
  if (status < 0)
    perror ("envoi de la donnee");
  close (sock);
  printf ("%s\n",data);
  )
}
)
```

ANNEXE 5 :

Listing des programmes constituant le projet concernant la configuration des vues "Localisation" :

- **Crea_Attr_Location_dans_Spectrum_1**
- **Crea_Attr_Location_dans_Spectrum_2**
- **Creation_Vues_Localisation_1**
- **Creation_Vues_Localisation_2**
- **Supp_Attr_Location_dans-Spectrum**
- **Suppression_Devices_Des_Vues_Location**

```
/* Ce programme recherche dans la BD oracle la localisation des devives     */
/* figurant dans la base de donnees de Spectrum.                            */
/* Tous les devices possedant une localisation sont copies dans le fichier  */
/* 'Modeles_Avec_Attr_Location'.                                            */


#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <math.h>


#define Max_Ent1 340                /* Longueur de la table des devices 'tbl_dev' */

#define USERNAME "csnet_query"      /* Username & Password pour interroger */
#define PASSWORD "shiva"            /* la BD ORACLE.                       */


EXEC SQL INCLUDE SQLCA;

EXEC SQL BEGIN DECLARE SECTION;
   char *username = USERNAME;
   char *password = PASSWORD;
   char loc[512];
   int ip_1_int;                /* Variables SQL */
   int ip_2_int;
   int ip_3_int;
   int ip_4_int;
EXEC SQL END DECLARE SECTION;


typedef struct table_device  /* Table des devices possedant une adresse IP */
 {
  char id_nom[9];
  char ip[30];
 };

struct table_device Dev_tbl[Max_Ent1];


main ()

{
 typedef enum BOOL (false,true) bool;

 char ligne[512],command[512],
      id_nom[15],localisation[100],
      ip_1[5],ip_2[5],ip_3[5],ip_4[5],ip[30],id_type[16];

 bool trouve;

 int i,j,k,nbr_dev;

 FILE *d,*g,*f,*h;


 system ("/home/suncoms/spectrum/vnmsh/connect"); /* connexion au SPECTROServeur */


/* CREATION DE LA TABLE 'Dev_tbl' (table des devices (GnSNMPDev)          */


system ("/home/suncoms/spectrum/vnmsh/show models | grep GnSNMPDev > all_models");

h = fopen ("Problemes.log","w");
if (h == NULL) printf ("Erreur d'ouverture du fichier 'Problemes.log'.");

f = fopen ("all_models","r");
```



```
if (f == NULL) printf ("Erreur d'ouverture du fichier 'all_models'.");

d = fopen ("Modeles_Avec_Attr_Location","w");
if (d == NULL) printf ("Erreur d'ouverture du fichier 'Modeles_Avec_Attr_Location'.");

nbr_dev = 0;
j = 0;
fgets(ligne,512,f);
while (fgets(ligne,512,f) != NULL)
 {
  for (i=0;i<=7;i++) id_nom[i] = ligne[i];
  id_nom[8] = '\0';
  i = 9;

  nbr_dev++;
  sprintf (command,"/home/suncoms/spectrum/vnmsh/show attributes attr=0x1027f mh=%s > va
leur_attribut",id_nom);
  system (command);

  g = fopen ("valeur_attribut","r");
  if (g == NULL) printf ("Erreur d'ouverture du fichier 'valeur_attribut'.");

  trouve = false;
  fgets(ligne,512,g);
  if (fgets(ligne,512,g) != NULL)
   {
    i = 45;
    k = 0;
    while (((ligne[i] != ' ') || (ligne[i+1] != ' ')) && (ligne[i] != '\n'))
     {
      ip[k] = ligne[i];
      i++; k++;
      trouve = true;
     }
    ip[k] = '\0';
   }
  fclose (g);
  if ((trouve == true) && (j <= Max_Ent1))
   {
    strcpy (Dev_tbl[j].id_nom,id_nom);
    strcpy (Dev_tbl[j].ip,ip);
    j++;
   }
  if (trouve == false)
   {
    fprintf (h,"Le modele %s n'a pas d'adresse IP\n",id_nom);
    printf ("Le modele %s n'a pas d'adresse IP\n",id_nom);
    nbr_dev--;
   }
 }

if (nbr_dev > Max_Ent1)
 {
  printf ("La longueur de la table des devices 'Dev_tbl' est trop petite.\n");
  printf ("Veuillez modifier la valeur de 'Max_Ent1'. Nouvelle valeur = %d.\n",nbr_dev);
 }
else printf ("\nTABLE 'Dev_tbl' de longueur %d CREEE.\n\n",nbr_dev);

fclose (f);



/* ECRITURE DANS LE FICHIER 'Modeles_Avec_Attr_Location' DES MODELES AVEC  */
/* UNE LOCALISATION.                                                       */


EXEC SQL CONNECT :username IDENTIFIED BY :password;
if (sqlca.sqlcode != 0) printf ("\nERREUR DE CONNECTION SUR ORACLE.\n");
```

```
else
 {
  printf ("Connected to ORACLE.\n");

  j = 0;
  while (j <= nbr_dev - 1)
   {
    k = 0;
    i = 0;
    while (Dev_tbl[j].ip[i] != '.')
     {                                  /* Assignation des variables SQL avec */
      ip_1[k] = Dev_tbl[j].ip[i];       /* l'adresse IP des devices.          */
      i++; k++;
     }
    ip_1[k] = '\0';
    ip_1_int = atoi(ip_1);
    k = 0;
    i++;
    while (Dev_tbl[j].ip[i] != '.')
     {
      ip_2[k] = Dev_tbl[j].ip[i];
      i++; k++;
     }
    ip_2[k] = '\0';
    ip_2_int = atoi(ip_2);
    k = 0;
    i++;
    while (Dev_tbl[j].ip[i] != '.')
     {
      ip_3[k] = Dev_tbl[j].ip[i];
      i++; k++;
     }
    ip_3[k] = '\0';
    ip_3_int = atoi(ip_3);
    k = 0;
    i++;
    while (Dev_tbl[j].ip[i] != '\0')
     {
      ip_4[k] = Dev_tbl[j].ip[i];
      i++; k++;
     }
    ip_4[k] = '\0';
    ip_4_int = atoi(ip_4);
    i++;
    k = 0;

                  /* Requete SQL: */

    EXEC SQL SELECT N.location
          INTO :loc
          FROM network_device N, tcpip_address T
          WHERE (T.IP1 = :ip_1_int and T.IP2 = :ip_2_int and T.IP3 = :ip_3_int and T.IP4
 = :ip_4_int)
          AND N.USUAL_NAME = T.USUAL_NAME;


    if (sqlca.sqlcode == -1405)
     {
      printf ("Le modele %s n'a pas de localisation dans ORACLE.\n",Dev_tbl[j]);
      fprintf (h,"Le modele %s n'a pas de localisation dans ORACLE.\n",Dev_tbl[j].id_nom
);
     }

    else if (sqlca.sqlcode == 1403)
     {
      fprintf (h,"Le modele %s a une adresse IP qui ne se trouve pas dans ORACLE.\n",Dev
_tbl[j].id_nom);
      printf ("Le modele %s a une adresse IP qui ne se trouve pas dans ORACLE.\n",Dev_tb
l[j].id_nom);
```



```
     }

        else if (sqlca.sqlcode == 0)
              {
               i = 0;
               k = 0;
               while (((loc[i] != ' ') || (loc[i+1] != ' ') || (loc[i+2] != ' ')) && (l
oc[i] != '\n'))
                {
                 localisation[k] = loc[i];
                 i++; k++;
                }
               localisation[k] = '\0';
               fprintf (d,"%s %s\n",Dev_tbl[j].id_nom,localisation);
              }
    j++;  /* passage au device suivant dans la table 'tbl_Dev'. */
   }
 }
 system ("/home/suncoms/spectrum/vnmsh/disconnect");
 fclose (h);
 fclose (d);
}
```

```
/* Ce programme initialise l'attribut correspondant a la localisation du    */
/* device avec la localisation trouvee dans le fichier                      */
/* 'Modeles_Avec_Attr_Location' cree par le programme                       */
/* 'Crea_Attr_Location_dans_Spectrum_1' et qui reprend tous les devices qui */
/* possedent une localisation dans la base de donnees ORACLE.               */


#include <ctype.h>
#include <string.h>
#include <math.h>
#include <stdlib.h>

#ifndef __CSCOREATRIDS_H__
#include "CsCoreAtrIds.h"
#endif

#include "CsAssocList.h"
#include "CsAssoc.h"
#include "CsBuffer.h"
#include "CsRelDef.h"
#include "CsSMailServ.h"
#include "CsSYMdlHnd.h"
#include "CsSYRelHnd.h"
#include "CsAttrDesc.h"
#include "CsAttrVal.h"
#include "CsAttrValL.h"
#include "SDemoDefs.h"
#include "CsVParmBlk.h"


/* PROCEDURE QUI TRANSFORME UN HEXADECIMAL EN DECIMAL     */


int atoX (char *ptr)

{
 int value = 0;
 while (isxdigit (*ptr))
  {
   value = 16*value + (*ptr<='9' ? *ptr-'0': (*ptr<='F' ? *ptr+10-'A':
                       *ptr+10-'a'));
   ++ptr;
  }
 return (value);
}


 CsSMailService *THE_mail_service = NULL;

 CsBuffer *buffer;

 CsModelHandle mh;

 void *data;

 char name[10],ligne[512],ident[8],location[50];

 char *vnm = name, *q = ident;

 int i,j,int_ident;

 FILE *f;


main ()
```



```
{
 j = 0;
 strcpy (name,"suncoms");

 f = fopen ("Modeles_Avec_Attr_Location","r");
 if (f == NULL) printf ("Erreur du fichier 'Modeles_Avec_Attr_Location'.\n");

 THE_mail_service = new CsSMailService (vnm , SS_PORT_ADDRESS);
 if ( THE_mail_service )
  {
   if ( !THE_mail_service->are_you_alive() )
    {
     cerr << "\n\nConnection failure with Spect" << endl;
     exit(EXIT_NO_SPECTROSERVER);
    };

   while (fgets(ligne,512,f) != NULL)
    {
     j++;
     printf ("j: %d.\n",j);
     for (i = 2; i < 8; i++) ident[i-2] = ligne[i];
     ident[6] = '\0';
     int_ident = atoX(q);
     i = 9;
     while (ligne[i] != '\n')
      {
       location[i-9] = ligne[i];
       i++;
      }
     location[i-9] = '\0';

     printf ("ident: %s, location: %s\n",ident,location);
     buffer = new (location, 0) CsBuffer;
     data = buffer;
     CsTulong attr = 0x23000d;

     CsSYModelHandle * uimh = new CsSYModelHandle( (CsTulong) int_ident );

     CsAttrValList * objet = new CsAttrValList ( 1 );

     objet->set_id ( 0, attr );

     objet->copy_val ( 0, attr, data, CsAttrDesc::TEXT_STRING );
     cout << "copy_val: " << objet->get_error(0) << endl;

     CsAttrValList * res = uimh->write_sync ( objet );
     cout << "write_sync: " << hex << res->get_error() << endl;
     cout << endl;
     delete (res);
     delete (uimh);
    }
  }
 fclose (f);
}
```

```
/* Ce programme recherche dans la BD oracle l'emplacement des devives       */
/* figurant dans la BD de Spectrum: le batiment ou la salle dans lequel il  */
/* est situe.                                                               */
/* Il cree un fichier reprenant tous les devices crees dans Spectrum        */
/* 'modeles_crees_dans_spectrum', un fichier reprenant tous les devices     */
/* devant etre crees dans des buildings 'modeles_crees_dans_buildings', un  */
/* fichier reprenant tous les devices devant etre crees dans des rooms      */
/* 'modeles_crees_dans_rooms' et un fichier indiquant le nombre de          */
/* devices dans chaque room 'Rooms' (sauf si le nombre de devices est 0).   */


#include <stdlib.h>
#include <string.h>
#include <stdio.h>

#define Max_Ent1 320      /* Longueur de la table 'Building_tbl'  */
#define Max_Ent2 220      /* Longueur de la table 'Room_table'    */
#define Max_Ent3 340      /* Longueur de la table 'Dev_table'     */

#define USERNAME "csnet_query"  /* Username & Password pour interroger */
#define PASSWORD "shiva"        /* la BD ORACLE.                       */


EXEC SQL INCLUDE SQLCA;

EXEC SQL BEGIN DECLARE SECTION;
   char *username = USERNAME;
   char *password = PASSWORD;
   char loc[50];
   int ip_1_int;
   int ip_2_int;                /* Variables SQL */
   int ip_3_int;
   int ip_4_int;
EXEC SQL END DECLARE SECTION;


typedef struct table
 {
  char id_nom[9];
  char nom[40];
  int nb_dev;
 };

typedef struct table_device
 {
  char id_nom[9];
  char nom[40];
  char ip[30];
  char id_type[9];
 };

struct table Room_tbl[Max_Ent2];
struct table Building_tbl[Max_Ent1];
struct table_device Dev_tbl[Max_Ent3];


main ()

{
 typedef enum BOOL {false,true} bool;

 char ligne[512],command[512],
      bat[40],optional[40],room[40],type[40],id_nom[15],nom[40],
      ip_1[5],ip_2[5],ip_3[5],ip_4[5],ip[30],id_type[16];

 bool trouve,salle,option,bonne_syntaxe_room,bonne_syntaxe_bat;

 int b,r,i,j,k,nbr_dev,nbr_buildings,nbr_rooms;
```



```
 FILE *q,*g,*d,*e,*eb,*er,*f,*h,*fopen();


 system ("/home/suncoms/spectrum/vnmsh/connect");


/* CREATION DE LA TABLE 'Building_tbl'                                      */


system ("/home/suncoms/spectrum/vnmsh/show models | grep Building > all_models");

f = fopen ("all_models","r");
if (f == NULL) printf ("Erreur d'ouverture du fichier 'all_models'.");
j = 0;

while (fgets(ligne,512,f) != NULL)
 {
  if (j <= Max_Ent1)
   {
    for (i=0;i<=7;i++) Building_tbl[j].id_nom[i] = ligne[i];
    Building_tbl[j].id_nom[8] = '\0';
    i = 9;
    while ((ligne[i] != ' ') || (ligne[i+1] != ' '))
     {
      Building_tbl[j].nom[i-9] = ligne[i];
      i++;
     }
    Building_tbl[j].nom[i-9] = '\0';
    Building_tbl[j].nb_dev = 0;
   }
  j++;
 }
nbr_buildings = j;
if (nbr_buildings > Max_Ent1)
 {
  printf ("La longueur de la table des buildings 'Building_tbl' est trop petite.\n");
  printf ("Veuillez modifier la valeur de 'Max_Ent1'. Nouvelle valeur = %d.\n",nbr_build
ings);
 }
else printf ("\nTABLE 'Building_tbl' de longueur %d CREEE.\n\n",nbr_buildings);
fclose (f);


/* CREATION DE LA TABLE 'Room_tbl'                                          */


system ("/home/suncoms/spectrum/vnmsh/show models | grep Room > all_models");

f = fopen ("all_models","r");
if (f == NULL) printf ("Erreur d'ouverture du fichier 'all_models'.");
j = 0;

while (fgets(ligne,512,f) != NULL)
 {
  if (j <= Max_Ent2)
   {
    for (i=0;i<=7;i++) Room_tbl[j].id_nom[i] = ligne[i];
    Room_tbl[j].id_nom[8] = '\0';
    i = 9;
    while ((ligne[i] != ' ') || (ligne[i+1] != ' '))
     {
      Room_tbl[j].nom[i-9] = ligne[i];
      i++;
     }
    Room_tbl[j].nom[i-9] = '\0';
    Room_tbl[j].nb_dev = 0;
   }
  j++;
```

```
 }
nbr_rooms = j;
if (nbr_rooms > Max_Ent2)
 {
  printf ("La longueur de la table des rooms 'Room_tbl' est trop petite.\n");
  printf ("Veuillez modifier la valeur de 'Max_Ent2'. Nouvelle valeur = %d.\n",nbr_rooms
);
 }
else printf ("\nTABLE 'Room_tbl' de longueur %d CREEE.\n\n",nbr_rooms);
fclose (f);



/* CREATION DE LA TABLE 'Dev_tbl'                                          */


system ("/home/suncoms/spectrum/vnmsh/show models > all_models");

h = fopen ("Problemes.log","w");
if (h == NULL) printf ("Erreur d'ouverture du fichier 'Problemes.log'.");

f = fopen ("all_models","r");
if (f == NULL) printf ("Erreur d'ouverture du fichier 'all_models'.");

d = fopen ("modeles_non_crees","w");
if (d == NULL) printf ("Erreur d'ouverture du fichier 'modeles_non_crees'.");

nbr_dev = 0;
j = 0;
fgets(ligne,512,f);
while (fgets(ligne,512,f) != NULL)
 {
  for (i=0;i<=7;i++) id_nom[i] = ligne[i];
  id_nom[8] = '\0';
  i = 9;
  while ((ligne[i] != ' ') || (ligne[i+1] != ' '))
   {
    nom[i-9] = ligne[i];
    i++;
   }
  nom[i-9] = '\0';
  while (ligne[i] == ' ') i++;
  for (k = 0; k < 8; k++) {id_type[k] = ligne[i]; i++;}
  id_type[8] = '\0';
  while (ligne[i] == ' ') i++;
  k = 0;
  while ((ligne[i] != ' ') && (ligne[i] != '\n'))
   {
    type[k] = ligne[i];
    i++; k++;
   }
  type[k] = '\0';

  if ((strcmp (type,"GnSNMPDev") == 0) || (strcmp (type,"Pingable") == 0) || (strncmp (t
ype,"Rtr_Shiva",9) == 0))
   {
    nbr_dev++;
    sprintf (command,"/home/suncoms/spectrum/vnmsh/show attributes attr=0x1027f mh=%s >
valeur_attribut",id_nom);
    system (command);

    g = fopen ("valeur_attribut","r");
    if (g == NULL) printf ("Erreur d'ouverture du fichier 'valeur_attribut'.");

    trouve = false;
    fgets(ligne,512,g);
    if (fgets(ligne,512,g) != NULL)
     {
```



```
      i = 45;
      k = 0;
      while (((ligne[i] != ' ') || (ligne[i+1] != ' ')) && (ligne[i] != '\n'))
       {
        ip[k] = ligne[i];
        i++; k++;
        trouve = true;
       }
      ip[k] = '\0';
     }
    fclose (g);
    if ((trouve == true) && (j <= Max_Ent3))
     {
      strcpy (Dev_tbl[j].nom,nom);
      strcpy (Dev_tbl[j].id_nom,id_nom);
      strcpy (Dev_tbl[j].ip,ip);
      strcpy (Dev_tbl[j].id_type,id_type);
      j++;
     }
    if (trouve == false)
     {
      fprintf (d,"Le modele %s (%s) n'a pas d'adresse IP\n",nom,id_nom);
      printf ("Le modele %s (%s) n'a pas d'adresse IP\n",nom,id_nom);
      nbr_dev--;
     }
   }
 }

if (nbr_dev > Max_Ent3)
 {
  printf ("La longueur de la table des devices 'Dev_tbl' est trop petite.\n");
  printf ("Veuillez modifier la valeur de 'Max_Ent3'. Nouvelle valeur = %d.\n",nbr_dev);
 }
else printf ("\nTABLE 'Dev_tbl' de longueur %d CREEE.\n\n",nbr_dev);

fclose (f);



/* CREATION DES DEVICES DANS LES 'Location_views' ADEQUATES                */


EXEC SQL CONNECT :username IDENTIFIED BY :password;
if (sqlca.sqlcode != 0) printf ("\nERREUR DE CONNECTION SUR ORACLE.\n");
else
 {
  printf ("Connected to ORACLE.\n");

  e = fopen ("modeles_crees_dans_spectrum","w");
  if (e == NULL) printf ("Erreur d'ouverture du fichier 'modeles_crees_dans_spectrum'.")
;

  eb = fopen ("modeles_crees_dans_buildings","w");
  if (eb == NULL) printf ("Erreur d'ouverture du fichier 'modeles_crees_dans_buildings'.
");

  er = fopen ("modeles_crees_dans_rooms","w");
  if (eb == NULL) printf ("Erreur d'ouverture du fichier 'modeles_crees_dans_rooms'.");

  j = 0;
  while (j <= nbr_dev - 1)
   {
    k = 0;
    i = 0;
    while (Dev_tbl[j].ip[i] != '.')
     {
      ip_1[k] = Dev_tbl[j].ip[i];
      i++; k++;
```

```
     }
    ip_1[k] = '\0';
    ip_1_int = atoi(ip_1);
    k = 0;
    i++;
    while (Dev_tbl[j].ip[i] != '.')
     {
      ip_2[k] = Dev_tbl[j].ip[i];
      i++; k++;
     }
    ip_2[k] = '\0';
    ip_2_int = atoi(ip_2);
    k = 0;
    i++;
    while (Dev_tbl[j].ip[i] != '.')
     {
      ip_3[k] = Dev_tbl[j].ip[i];       /* Initialisation des variables SQL */
      i++; k++;
     }
    ip_3[k] = '\0';
    ip_3_int = atoi(ip_3);
    k = 0;
    i++;
    while (Dev_tbl[j].ip[i] != '\0')
     {
      ip_4[k] = Dev_tbl[j].ip[i];
      i++; k++;
     }
    ip_4[k] = '\0';
    ip_4_int = atoi(ip_4);
    i++;
    k = 0;

    printf ("\n ---> %s  %s %d.%d.%d.%d\n",Dev_tbl[j].id_nom,Dev_tbl[j].nom,ip_1_int,ip_
2_int,ip_3_int,ip_4_int);


             /* Requete SQL */

    EXEC SQL SELECT N.location
          INTO :loc
          FROM network_device N, tcpip_address T
          WHERE (T.IP1 = :ip_1_int and T.IP2 = :ip_2_int and T.IP3 = :ip_3_int and T.IP4
 = :ip_4_int)
          AND N.USUAL_NAME = T.USUAL_NAME;


    if (sqlca.sqlcode == -1405)
     {
      printf ("Le modele %s n'a pas de localisation dans ORACLE.\n",Dev_tbl[j]);
      fprintf (d,"%s: Le modele %s n'a pas de localisation dans ORACLE.\n",Dev_tbl[j].id
_nom,Dev_tbl[j].nom);
     }
    else if (sqlca.sqlcode == 1403)
          {
           fprintf (d,"Le modele %s (%s) a une adresse IP qui ne se trouve pas dans ORAC
LE.\n", Dev_tbl[j].nom,Dev_tbl[j].id_nom);
           printf ("Le modele %s (%s) a une adresse IP qui ne se trouve pas dans ORACLE.
\n", Dev_tbl[j].nom,Dev_tbl[j].id_nom);
          }
     else if (sqlca.sqlcode == 0)
          {
           k = 0;
           i = 0;
           if ((loc[i] >= '0') && (loc[i] <= '9'))
            {
             bonne_syntaxe_bat = true;
             while ((loc[i] != ' ') && (loc[i] != '\n'))
              {
```



```
               bat[k] = loc[i];
               i++; k++;
              }
             bat[k] = '\0';
            }
           else
            {
             bonne_syntaxe_bat = false;
             while (((loc[i] != ' ') || (loc[i+1] != ' ')) && (loc[i] != '\n'))
              {
               bat[k] = loc[i];
               i++; k++;
              }
             bat[k] = '\0';
            }
           b = 0;
           trouve = false;
           if (bonne_syntaxe_bat == true)
            {
             while ((b <= nbr_buildings - 1) && (trouve == false))
              {
               if (strcmp(bat,Building_tbl[b].nom) == 0) trouve = true;
               b++;
              }
             b--;
            }
           if (trouve == false)
            {
             if (bonne_syntaxe_bat == true)
              {
fprintf (d,"%s: Le modele %s est situe dans le batiment %s qui n'existe pas dans Spectru
m.\n"
                         ,Dev_tbl[j].id_nom,Dev_tbl[j].nom,bat);
printf ("Le modele %s est situe dans le batiment %s qui n'existe pas dans Spectrum.\n",D
ev_tbl[j].nom,bat);
              }
             else
              {
printf ("Le modele %s (IP: %s) est situe dans la 'location' %s qui n'a pas la bonne synt
axe dans Oracle.\n"
                         ,Dev_tbl[j].nom,Dev_tbl[j].ip,bat);
fprintf (d,"%s: Le modele %s est situe dans le batiment %s qui n'a pas la bonne syntaxe
dans Oracle.\n"
                         ,Dev_tbl[j].id_nom,Dev_tbl[j].nom,bat);
              }
            }
           else
            {
             salle = false;
             option = false;
             i++;
             if ((loc[i] == 'R') || (loc[i] == 'S') || ((loc[i] >= '0') && (loc[i] <= '
5')) && (loc[i+1] == '-'))
              {
               bonne_syntaxe_room = true;
               i = i+2;
               salle = true;
               k = 0;
               while ((loc[i] != ' ') && (loc[i] != '\n'))
                {
                 room[k] = toupper(loc[i]);
                 i++; k++;
                }
               room[k] = '\0';
              }
             else
              {
               if (loc[i-1] != '\0') bonne_syntaxe_room = false;
                {
```

```
                k = 0;
                while ((loc[i] != '\0') && (loc[i] != '\n'))
                 {
                  room[k] = toupper(loc[i]);
                  i++; k++;
                 }
                room[k] = '\0';
               }
             }
            if (bonne_syntaxe_room == false)
             {
printf ("Le modele %s (IP: %s) est situe dans la 'location' qui n'a pas la bonne syntaxe
 pour la room %s dans Oracle.\n"
                    ,Dev_tbl[j].nom,Dev_tbl[j].ip,room);
fprintf (d,"%s: Le modele %s est situe dans la room %s qui n'a pas la bonne syntaxe dans
 Oracle.\n"
                    ,Dev_tbl[j].id_nom,Dev_tbl[j].nom,room);
             }
            while ((loc[i] != ':') && (i<59)) i++;
            if (i < 59)
             {
              i = i+2;
              option = true;
              k = 0;
              while (((loc[i] != ' ') || (loc[i+1] != ' ')) && (loc[i] != '\n'))
               {
                optional[k] = toupper(loc[i]);
                i++; k++;
               }
              optional[k] = '\0';
             }
            if ((salle == true) || (option == true))
             {
              r = 0;
              trouve = false;
              while ((r <= nbr_rooms - 1) && (trouve == false))
               {
                if (option == true) if (strncmp(optional,Room_tbl[r].nom,8) == 0) trou
ve = true;
                else if (strcmp(room,Room_tbl[r].nom) == 0) trouve = true;
                r++;
               }
              r--;
              if (trouve == false)
               {
                Building_tbl[b].nb_dev++;
                fprintf (e,"%s %s %s %s.\n",Building_tbl[b].id_nom,Dev_tbl[j].id_nom,D
ev_tbl[j].nom,bat);
                fprintf (eb,"%s %s %s %s %s.\n",Building_tbl[b].id_nom,Dev_tbl[j].id_n
om,Dev_tbl[j].id_type,Dev_tbl[j].nom,bat);
                sprintf (command,"/home/suncoms/spectrum/vnmsh/create association rel=
Contains lmh=%s rmh=%s",
                         Building_tbl[b].id_nom,Dev_tbl[j].id_nom);
                system (command);
                fprintf (h,"%s: La room no %s dans le batiment no %s n'existe pas dans
 Spectrum.\n",
                         Dev_tbl[j].nom,Room_tbl[r].nom,bat);
               }
              else
               {
                Room_tbl[r].nb_dev++;
                fprintf (e,"%s %s %s %s.\n",Room_tbl[r].id_nom,Dev_tbl[j].id_nom,Dev_t
bl[j].nom,bat);
                fprintf (er,"%s %s %s %s %s.\n",Room_tbl[r].id_nom,Dev_tbl[j].id_nom,D
ev_tbl[j].id_type,
                         Dev_tbl[j].nom,Room_tbl[r].nom);
                sprintf (command,"/home/suncoms/spectrum/vnmsh/create association rel=
Contains lmh=%s rmh=%s",
                         Room_tbl[r].id_nom,Dev_tbl[j].id_nom);
```



```
                system (command);
               }
              }
             else
              {
               Building_tbl[b].nb_dev++;
               fprintf (e,"%s %s %s %s.\n",Building_tbl[b].id_nom,Dev_tbl[j].id_nom,Dev
_tbl[j].nom,bat);
               fprintf (eb,"%s %s %s %s %s.\n",Building_tbl[b].id_nom,Dev_tbl[j].id_nom
,Dev_tbl[j].id_type,Dev_tbl[j].nom,bat);
               sprintf (command,"/home/suncoms/spectrum/vnmsh/create association rel=Co
ntains lmh=%s rmh=%s",
                        Building_tbl[b].id_nom,Dev_tbl[j].id_nom);
               system (command);
              }
             }
           }
    j++;  /* Pour passer au device suivant dans la table 'Dev_tbl' */
   }
  fclose (d);
  fclose (er);
  fclose (e);
  fclose (eb);
  fclose (h);
 }
system ("/home/suncoms/spectrum/vnmsh/disconnect");


/* CREATION D'UN FICHIER CONTENANT LES ROOMS DANS LESQUELLES DOIVENT ETRE   */
/* PLACES DES DEVICES.                                                      */

q = fopen ("Rooms","w");
if (q == NULL) printf ("Erreur d'ouverture du fichier 'Rooms'.");

r = 0;
while (r <= nbr_rooms - 1)
 {
  if (Room_tbl[r].nb_dev != 0)
    fprintf (q,"%s %d\n",Room_tbl[r].id_nom,Room_tbl[r].nb_dev);
  r++;
 }
fclose (q);
}
```

```
/* Ce programme place les devices contenus dans les fichiers                */
/* 'modeles_crees_dans_rooms' dans les rooms adequats.
   */
/* Il les dispose de maniere a ne pas avoir tous les devices au meme        */
/* endroit dans la fenetre.                                                 */
/* Il calcule leur position dans la fenetre en fonction du nombre de        */
/* devices devant etre disposes dans la fenetre. Ces nombres sont contenus  */
/* dans le fichier 'Rooms'.                                                 */



#include <ctype.h>
#include <string.h>
#include <math.h>
#include <stdlib.h>

#ifndef __CSCOREATRIDS_H__
#include "CsCoreAtrIds.h"
#endif

#include "CsAssocList.h"
#include "CsAssoc.h"
#include "CsBuffer.h"
#include "CsRelDef.h"
#include "CsSMailServ.h"
#include "CsSYMdlHnd.h"
#include "CsSYRelHnd.h"
#include "CsAttrDesc.h"
#include "CsAttrVal.h"
#include "CsAttrValL.h"
#include "SDemoDefs.h"
#include "CsVParmBlk.h"

#define debut_d_une_ligne 90          /* postion X en pixels du 1er device */
#define debut_d_une_colonne 90        /* postion Y en pixels du 1er device */
#define espacement 120                /* espacement entre 2 devices en pixels */
#define nbr_device_par_ligne 6  /* nombre de devices places sur 1 meme ligne */


 CsTulong posX,posY;

 CsTuchar *resX, *resY;
 char buf2[36];
 int l;


/* PROCEDURE QUI TRANSFORME UN STRING (REPRESENTANT UN NOMBRE HEXADECIMAL)  */
/* EN UN ENTIER MIS SOUS FORME HEXADECIMAL.                                 */

int atoX (char *ptr)

{
 int value = 0;
 while (isxdigit (*ptr))
  {
   value = 16*value + (*ptr<='9' ? *ptr-'0': (*ptr<='F' ? *ptr+10-'A':
                       *ptr+10-'a'));
   ++ptr;
  }
 return (value);
}



/* PROCEDURE QUI INVERSE UN STRING 2 A 2.  (4008FF  -> FF.8.40.0)           */

void inverse (char a[6], char b[18])

{
```



```
 if ((a[2] == '0') && (a[4] == '0'))
  sprintf (b,".%c.%c.%c%c.0",a[5],a[3],a[0],a[1]);
 else if (a[4] == '0')
       sprintf (b,".%c.%c%c.%c%c.0",a[5],a[2],a[3],a[0],a[1]);
      else if (a[2] == '0')
            sprintf (b,".%c%c.%c.%c%c.0",a[4],a[5],a[3],a[0],a[1]);
           else sprintf (b,".%c%c.%c%c.%c%c.0",a[4],a[5],a[2],a[3],a[0],a[1]);
}


/* PROCEDURE QUI CALCULE LA POSITION X (EN PIXELS) D'UN DEVICE EN FONCTION */
/* DE SON NUMERO DANS LA LIGNE.                                             */

void calcul_positionX (int i, char bufX[])

{
 posX = debut_d_une_ligne + (espacement * i);
 resX = (CsTuchar *)&posX;
 l = 4;
 while (l > 0)
  {
   l--;
   sprintf (bufX,".%x", (int) resX[l]);
   while (*bufX++);  --bufX;
  }
}


/* PROCEDURE QUI CALCULE LA POSITION X (EN PIXELS) D'UN DEVICE EN FONCTION */
/* DE SON NUMERO DANS LA LIGNE.                                             */

void calcul_positionY (int i, char bufY[])

{
 posY = debut_d_une_colonne + (espacement * i);
 resY = (CsTuchar *)&posY;
 l = 4;
 while (l > 0)
  {
   l--;
   sprintf (bufY,".%x", (int) resY[l]);
   while (*bufY++);  --bufY;
  }
}



main ()

{
 typedef enum BOOL (false,true) bool;

 char name[10],ligne[512],ligne_2[512],id_room[9],buf[1024],buf3[1024],
      nbr_dev[4],id_room_2[6],id_dev_2[6],id_type_2[6],buf1[18];

 char *p = id_room;

 CsBuffer *buffer;

 CsModelHandle mh;

 CsSMailService *THE_mail_service = NULL;

 void *data;

 int i,j,int_nbr_dev,compt,colonne,n_ligne;

 FILE *f,*g;
```

```
/* Ce programme efface la valeur qui se trouve dans le champ "Location"     */
/* de la fenetre reprenant les informations sur un modele.                  */
/* Il utilise le fichier 'Modeles_Avec_Attr_Location' qui est cree par      */
/* le programme 'Crea_Attr_Location_dans_Spectrum_1' et qui reprend tous les*/
/* modeles qui possedent une localisation dans la base de donnees ORACLE.   */



#include <ctype.h>
#include <string.h>
#include <math.h>
#include <stdlib.h>

#ifndef __CSCOREATRIDS_H__
#include "CsCoreAtrIds.h"
#endif

#include "CsAssocList.h"
#include "CsAssoc.h"
#include "CsBuffer.h"
#include "CsRelDef.h"
#include "CsSMailServ.h"
#include "CsSYMdlHnd.h"
#include "CsSYRelHnd.h"
#include "CsAttrDesc.h"
#include "CsAttrVal.h"
#include "CsAttrValL.h"
#include "SDemoDefs.h"
#include "CsVParmBlk.h"


/* PROCEDURE QUI TRANSFORME UN STRING REPRESENTANT UN NOMBRE SOUS FORME     */
/* HEXADECIMAL EN UN ENTIER.                                                */


int atoX (char *ptr)

{
 int value = 0;
 while (isxdigit (*ptr))
  {
   value = 16*value + (*ptr<='9' ? *ptr-'0': (*ptr<='F' ? *ptr+10-'A':
                       *ptr+10-'a'));
   ++ptr;
  }
 return (value);
}


 CsSMailService *THE_mail_service = NULL;
 CsBuffer *buffer;
 char name[10],ligne[512],ident[8],location[50];
 CsModelHandle mh;
 void *data;
 int i,j,int_ident;
 char *vnm = name, *q = ident;

 FILE *f;


main ()

{
 j = 0;
 strcpy (name,"suncoms");

 f = fopen ("Modeles_Avec_Attr_Location","r");
```



```
 if (f == NULL) printf ("Erreur du fichier 'Modeles_Avec_Attr_Location'.\n");

 THE_mail_service = new CsSMailService (vnm , SS_PORT_ADDRESS);
 if ( THE_mail_service )
  {
   if ( !THE_mail_service->are_you_alive() )
    {
     cerr << "\n\nConnection failure with Spect" << endl;
     exit(EXIT_NO_SPECTROSERVER);
    };

   while (fgets(ligne,512,f) != NULL)
    {
     j++;
     printf ("%d:\n",j);
     for (i = 2; i < 8; i++) ident[i-2] = ligne[i];
     ident[6] = '\0';
     int_ident = atoX(q);
     strcpy (location," ");

     printf ("ident: %s, location: %s\n",ident,location);
     buffer = new (location, 0) CsBuffer;
     data = buffer;
     CsTulong attr = 0x23000d;

     CsSYModelHandle * uimh = new CsSYModelHandle( (CsTulong) int_ident );

     CsAttrValList * objet = new CsAttrValList ( 1 );

     objet->set_id ( 0, attr );

     objet->copy_val ( 0, attr, data, CsAttrDesc::TEXT_STRING );
     cout << "copy_val: " << objet->get_error(0) << endl;

     CsAttrValList * res = uimh->write_sync ( objet );
     cout << "write_sync: " << hex << res->get_error() << endl;
     cout << endl;
     delete (res);
     delete (uimh);
    }
  }
 fclose (f);
}
```

```
g = fopen ("Rooms","r");
if (g == NULL) printf ("Erreur du fichier 'Rooms'.\n");

j = 0;
strcpy (name,"suncoms");

THE_mail_service = new CsSMailService (name , SS_PORT_ADDRESS);
if ( THE_mail_service )
 {
  if ( !THE_mail_service->are_you_alive() )
   {
    cerr << "\n\nConnection failure with Spect" << endl;
    exit(EXIT_NO_SPECTROSERVER);
   };

  while (fgets (ligne,512,g) != NULL)
   {
    for (i = 2; i < 8; i++) id_room[i-2] = ligne[i];
    id_room[6] = '\0';
    mh = atoX (p);
    i = 9;
    while (ligne[i] != '\n')
     {
      nbr_dev[i-9] = ligne[i];
      i++;
     }
    nbr_dev[i-9] = '\0';
    int_nbr_dev = atoi (nbr_dev);

    f = fopen ("modeles_crees_dans_rooms","r");
    if (f == NULL) printf ("Erreur du fichier 'modeles_crees_dans_rooms'.\n");

    sprintf (buf,"%x.0.0.0",int_nbr_dev);
    compt = 1;
    colonne = 0;
    n_ligne = 0;
    while ((fgets (ligne_2,512,f) != NULL) && (compt <= int_nbr_dev))
     {
      for (i = 2; i < 8; i++) id_room_2[i-2] = ligne_2[i];
      id_room_2[6] = '\0';
      if (strcmp(id_room,id_room_2) == 0)
       {
        for (i = 11; i < 17; i++) id_dev_2[i-11] = ligne_2[i];
        id_dev_2[6] = '\0';
        i = 20;
        while (ligne_2[i] != ' ')
         {
          id_type_2[i-20] = ligne_2[i];
          i++;
         }
        if (i == 25) id_type_2[5] = '0';
        id_type_2[6] = '\0';
        inverse (id_dev_2,buf1);
        strcat (buf,buf1);
        inverse (id_type_2,buf1);
        strcat (buf,buf1);
        strcat (buf,".0.0.0.0");
        if ((fmod (colonne,nbr_device_par_ligne) == 0) && (colonne != 0))
         {
          colonne = 0;
          n_ligne++;
         }
        calcul_positionX (colonne,buf2);
        strcat (buf,buf2);
        calcul_positionY (n_ligne,buf2);
        strcat (buf,buf2);

        compt++;
```



```
        colonne++;
       }
     }
    fclose (f);
    i = 0;
    while (buf[i] != '\0')
     {
      buf3[i] = toupper (buf[i]);
      i++;
     }
    buf3[i] = '\0';
    cout << "Identifiant de la room: " << id_room << endl;
    cout << endl;
    cout <<  "BUFFER: " << buf3 << endl;

    buffer = new (buf3, 16) CsBuffer;
    data = buffer;
    CsTulong attr = 0x10037;
    CsSYModelHandle * uimh = new CsSYModelHandle ( mh );
    CsAttrValList * objet = new CsAttrValList ( 1 );
    objet->copy_val ( 0, attr, data, CsAttrDesc::OCTET_STRING );
    CsAttrValList * res = uimh->write_sync ( objet );
    cout << "write_sync: " << hex << res->get_error(0) << endl;
    cout << endl;
    delete (uimh);
   }
  fclose (g);
 }
}
```

```
/* Ce programme efface tous les modeles qui ont ete crees par le        */
/* programme "Creation_Vues_Locations"                                  */
/* Il travaille a partir du fichier 'modele_crees_dans_spectrum'.       */



#include <string.h>
#include <stdio.h>



main ()

{
typedef enum BOOL (false,true) bool;

char ligne[512],id_loc[9],id_nom[9],command[512];

int i;

FILE *f,*fopen();


 f = fopen ("modeles_crees_dans_spectrum","r");
 if (f == NULL) printf ("Erreur d'ouverture du Fichier 'modeles_crees_dans_spectrum'\n")
;

 system ("/home/suncoms/spectrum/vnmsh/connect");

 while (fgets(ligne,512,f) != NULL)
  {
   for (i=0;i<=7;i++) id_loc[i] = ligne[i];
   id_loc[8] = '\0';
   for (i=9;i<=16;i++) id_nom[i-9] = ligne[i];
   id_nom[8] = '\0';
   sprintf (command,"/home/suncoms/spectrum/vnmsh/destroy association -n rel=Contains lm
h=%s rmh=%s",id_loc,id_nom);
   system (command);
  }

 system ("/home/suncoms/spectrum/vnmsh/disconnect");

 fclose (f);
}
```